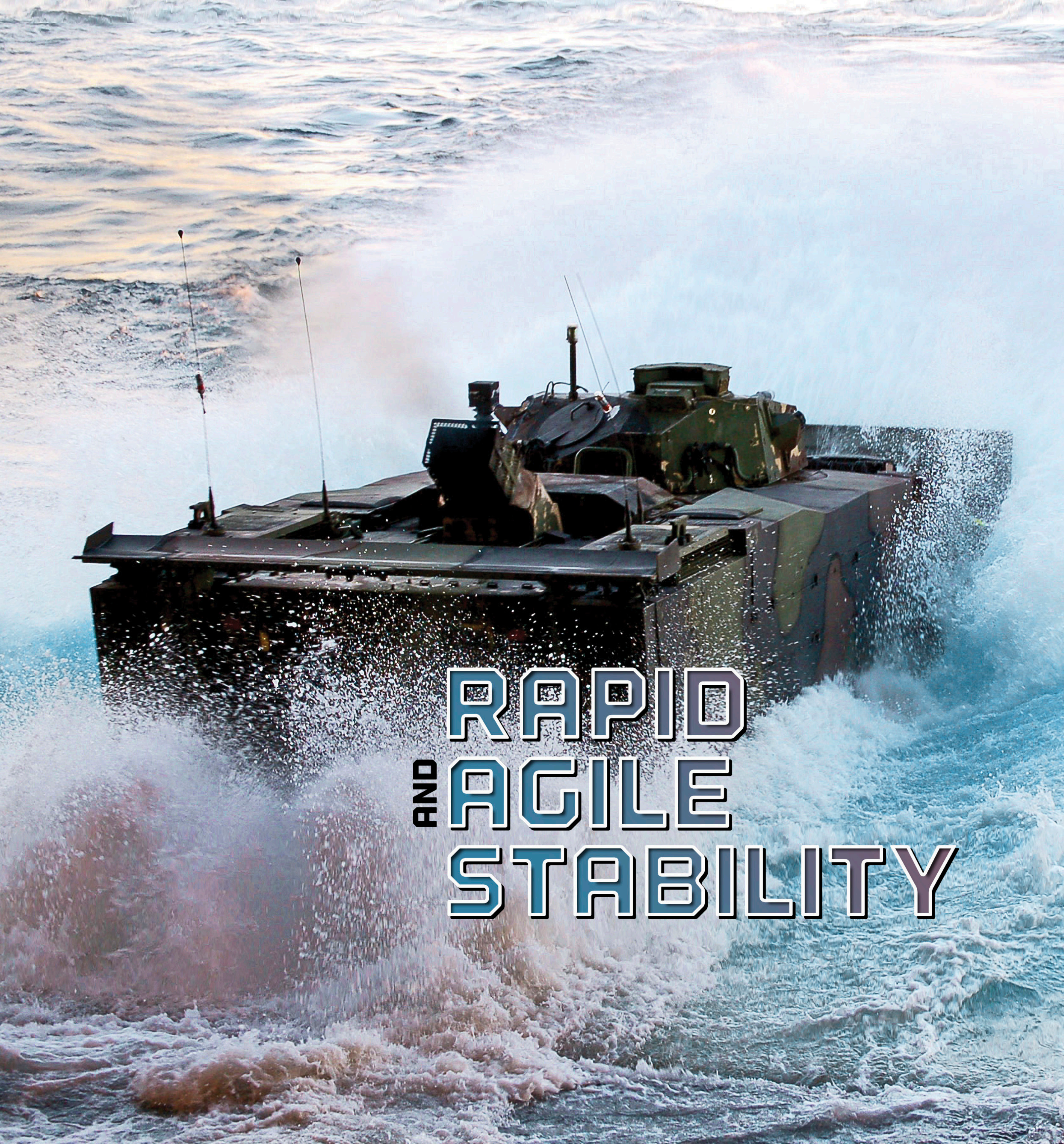
CrossTalk

May / June 2012 **The Journal of Defense Software Engineering** Vol. 25 No. 3



Rapid and Agile Stability

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **JUN 2012**	2. REPORT TYPE	3. DATES COVERED **00-05-2012 to 00-06-2012**

4. TITLE AND SUBTITLE **CrossTalk, The Journal of Defense Software Engineering. Volume 25, Number 3. May/June 2012**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave Bldg 1238,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**40**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

RAPID AND AGILE STABILITY

Departments

CrossTalk
May/June 2012 The Journal of Defense Software Engineering Vol. 25 No. 3
RAPID AND AGILE STABILITY

Cover Design by
Kent Bingham

Rapid and Agile Stability

CrossTalk

NAVAIR	Jeff Schwalb
DHS	Joe Jarzombek
309 SMXG	Karl Rogers

Publisher	Justin T. Hill
Advisor	Kasey Thompson
Article Coordinator	Lynne Wade
Managing Director	Tracy Stauder
Managing Editor	Brandon Ellis
Associate Editor	Colin Kelly
Art Director	Kevin Kiernan

Phone	801-775-5555
E-mail	stsc.customerservice@hill.af.mil
CrossTalk Online	www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: CrossTalk is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the CrossTalk webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>.
ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank NAVAIR for sponsoring this issue.

Rapid and Agile Stability

The family of software methodologies known as agile is a group of iterative and incremental development approaches where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. In all cases the goal is for teams to have the ability to respond to change while following an agreed-to plan, and applying an iterative approach in order to provide customers with early and continuous delivery of software. The stability is contained in teams that carry this out in a repeatable way based on a core set of principles.

While there is no single agile method, there are several methods that are influenced by agile principles in various ways. These span the spectrum from Extreme Programming to Scrum to TSP as well as others. In the end, the methods used to implement the process must be a good fit for the environment and the quality of the products required. Factors that influence the applicability of this "right fit" include size, complexity, safety, quality, cost, and schedule.

Agile principles may apply to projects that have development cycles that go anywhere from a few months to two or more years. Other factors that must be taken into account include the ability of an organization to adopt agile practices; the suitability of agile practices for a given product; and the suitability of agile practices for the organization responsible for product development. Agile principles embrace requirements changes, enable delivery of software frequently, emphasize the need for daily interaction between customers and developers, and value self-organizing teams. Agile processes provide sustainable development, call attention to the need for technical excellence and good design, and view simplicity as essential. In this environment, leadership must establish a regular drumbeat of face-to-face conversations with development teams to plan and track project status. To be effective at planning, tracking, and improving, these teams collect basic measures that reveal status during a project cycle and opportunities for improved future planning and other aspects of the methodologies being applied. Finally, these teams must decide together the priority of the products to be delivered according to the development environment and the needs of the sponsor and the user.

It is the attention to product quality plus the application of repeatable development methods that leads us back to the focus of this issue on Rapid and Agile Stability. I invite you to consider effective and repeatable software system development and maintenance approaches in both an agile and systematic way as you enjoy the articles in this month's issue of CrossTalk magazine.

Ms. Joan Johnson
Director, Software Engineering
Naval Air Systems Command

Individual Effort Estimating

Not Just for Teams Anymore

Russell Thackston, Auburn University
David A. Umphress, Auburn University

Abstract. Truly viable software—mobile device apps, services, components—are being written by one-person teams, thus demonstrating the need for engineering discipline at the individual level. This article examines effort estimation for individuals and proposes a lightweight approach based on a synthesis of a number of concepts derived from existing team estimation practices.

Software engineering is usually portrayed as a team activity, one where a myriad of technical players are choreographed to build a software solution to a complex problem. With such a perspective, estimating the effort required to write software involves a top-down view of development projects. Effort is forecasted based on a characterization of previous projects that is intended to represent teams performing to the statistical norm. While large projects are predominant—especially in the DoD environment—the fact remains that teams are made of unique individuals, each of whom write software at a different tempo and with different properties. At some point in a project, the top-down prediction of required effort must be tempered with a bottom-up frame of reference in which the team fades from being a group of generic members to being a collection of distinct persons.

Historically, estimation methods have focused on effort at the team level. Recent agile software development practices have shed light on taking individuals—who are acting as part of a team—into consideration. Emerging trends in software development for mobile devices suggest effort estimation methods can be employed for one-person endeavors and those methods can benefit teams, but those methods are still very much in their infancy. This paper examines effort estimation from the one-person team perspective and describes a lightweight effort estimation technique in that light.

Rise of the One-person Team

One-person software companies—historically referred to as shareware authors, but more recently labeled "micro Independent Software Vendors," or microISVs—are on the rise, fueled in part by the proliferation of free and open source tools and new ecosystems, such as Apple's App Store and the Android Market. In fact, metrics gathered by AdMob, a mobile advertising network, estimated the market for paid mobile apps in August of 2009 at $203 million [1]. That is an estimated $2.4 billion per year, for mobile apps alone, where the greatest barrier to entry is the cost of the mobile device.

For example, publishing a game in the Apple App Store requires an investment of around $100—not including a device—as the only requirement is the $99 per year to join the Apple iOS Developer Program [2]. Publishing an Android app is even less costly since the developer membership fee is only $25 [3]. Both platforms boast generous support documentation, which is freely available on the Apple and Google websites, respectively. Furthermore, many books are available for app authors wishing to dive deeper into the technologies of these platforms.

In addition to the mobile app market, opportunities exist in other markets, such as the traditional downloadable software venue as well as the relatively new concept of Software as a Service (SaaS) websites. Gartner Research measured worldwide SaaS revenue for 2010 at $10 billion and should reach $12.1 billion in 2011 [4].

MicroISVs encapsulate the entire spectrum of company activities into a single individual. As a result, the successful microISV focuses only on those activities that provide clear, measurable benefit and contribute directly to the bottom line; all other activities are deemed unimportant and are, as a result, discarded. It is through this natural selection process that successful microISV founders learn what is essential to the operation of a business and what is not. Unfortunately, the benefits derived from certain activities may not be directly visible. Many business activities—such as planning and risk management—are proven to provide measurable benefit, but may seem unimportant to a microISV operator confronted with the daily operations of a business. Too often, this category includes non-technical activities, such as taking the time to understand the size and scale of upcoming work (e.g. estimating effort).

The Plight of the Individual Estimator

MicroISVs are not the only one-person software development operations in action. Regardless of the size of the team (enterprise, company, firm, etc.), software development still boils down to the individual on the front line—the developer. No matter what label is given to the developer (microISV owner, consultant, or team member), it is his or her responsibility to help craft estimates and to manage his or her own time. The motivation varies, depending on the circumstances. For example, microISV owners might primarily use estimates to plan release cycles and orchestrate business and technical tasks. Consultants might use personal estimates to provide billing estimates. Team members could use personal estimates to validate requested delivery dates and coordinate time allocations among multiple activities and projects. However, individual software engineers remain generally ill equipped to construct personal estimates. This leaves the individual developer faced with the choice between guesswork, formal models, and/or relying on team-oriented techniques.

Unfortunately, team members, too, fall into behaviors which either avoid giving proper attention to estimating effort, or they fall back on one of the least accurate approaches: "gut feel" estimates.

Why do individual software developers not value good effort estimates? An informal survey of members of the Association of Software Professionals (ASP) reveals the perception that developing an estimate provides no direct value, either to microISV owners or their customers, the primary reason being that microISV product requirements are too fluid and are, therefore, not a good basis for an estimate [5]. Some microISV owners indicate that effort estimates are unnecessary unless a customer or client is holding them accountable; however, this is not often the case for microISVs, which tend to focus on shrink-wrapped products.

Many of the same arguments expressed by microISVs apply to individual developers acting as team members. Developers on a team may not be asked for effort estimates; someone in a senior position may directly provide deadlines to them. In the event that a developer is asked for an estimate, it is likely they are not properly equipped to provide a good estimate or they do not view this non-technical activity as an interesting and engaging problem, like coding. This can lead to an estimating rule of thumb such as, "Make a guess and multiply by three."

Regardless of the circumstances and rationale, many individual developers are not equipped to construct and derive benefit from personal effort estimates. The spectrum of effort estimation approaches is broad, often overwhelmingly so. At the near end of the spectrum lies guesswork; at the far end lies formal models. The former is quick, but difficult to tune; the latter involves complex mathematical calculations to model past performance and requires a heavy investment of time. The agile software development movement strives to strike a conciliatory balance that takes advantage of individual expert opinions adjusted by the collective wisdom of multiple participants. Planning Poker [6], for example, exemplifies this philosophy of guessing effort individually then attenuating the range of guesses through team dialog. Up to this point, there are no single person equivalents to Planning Poker (i.e., "Planning Solitaire").

Additionally, developers do not see a benefit to themselves in constructing an effort estimate. This is due in part to the tools and techniques currently available to the individual, which are either too heavyweight or non-existent. Either the process of constructing the estimate takes too much time and effort (heavyweight) or it produces low-quality results (lightweight or guessing). In either case, the ROI does not fit the circumstances. What is needed is a lightweight effort estimating process, focused on the individual, capable of constructing an estimate with a reasonable degree of accuracy.

Estimation Landscape

The process of estimating the cost of future software development efforts, in terms of time and resources, is a complex issue. Researchers have designed and tested models for predicting effort using a variety of approaches, techniques, and technologies. For example, while researchers found half of all published research on effort estimation (up through 2004) utilized some form of regression technique—predicting future effort based on past effort—a good deal of research was still going into other techniques such as function point analysis, expert judgment, and analogy [7]. These models are capable of predicting effort in a variety of environments with varying degrees of success and accuracy.

Most approaches share a common thread of complex mathematical models, requiring calibration and tuning. For example, the Constructive Cost Model (COCOMO), one of the most well known models, uses 15 cost drivers or "attributes" to estimate the size of the product to be created [8]. Organizations must accurately rate each cost driver, as well as determine software project coefficients that characterize their environment. Function point analysis employs a model based on determining the number and complexity of files internal to the system, files external to the system, inputs, outputs, and queries. This allows an organization to estimate effort by comparing the number and type of function points to historical data from past projects. Like COCOMO, function point analysis requires adjusting the estimate based on a variety of technical and operational characteristics, such as performance and reusability.

Despite the large amount of research focused on producing an accurate effort estimate, the typical software project is on time, on budget, and fully functional only about one-third of the time [9]. Clearly, the factors behind this statistic are poorly enumerated and vaguely understood, at best, given the number of variables involved. Despite the complexity of the problems facing the software cost estimating discipline, a wide variety of research into the problem has been conducted, focusing on such aspects as estimating approaches and models [10]. Perhaps one of the contributing factors lies at the bottom of the hierarchy, with software engineers who are ill equipped to provide estimates and validate deadlines imposed by their managers.

While there are many approaches to estimating effort, it is clear that the available approaches focus on the team or enterprise. Few approaches deal with individual effort estimation, such as at the task level. Furthermore, few approaches can be characterized as "lightweight" and suitable to the agile environments of one-person software development teams.

Estimation for the One-person Team

In 1995, Watts Humphrey introduced the Personal Software ProcessSM (PSPSM), which defined the first published software process tailored for the individual [11]. PSP defines a highly structured approach to measuring specific aspects of an individual's software development work. With respect to effort estimation, PSP employs a proxy-based approach referred to as Proxy-based Estimating (PROBE). In general, a proxy is a generic stand-in for something else, where the two objects' natures are similar. With regard to software estimating, a proxy is a completed task of similar size to an incomplete task. Therefore, it can be assumed the time to complete the second task should be similar to that of the first. PROBE specifically uses a lengthy, mathematical process for determining anticipated SLOC, which are used to compare tasks and find an appropriate proxy from which a time estimate is derived.

PSP has demonstrated popularity in both academic and business environments. Businesses, such as Motorola and Union Switch &

Signal, Inc., have adopted PSP and claimed varying degrees of success. Many universities have integrated PSP into software engineering courses in an effort to demonstrate and measure the value of a structured personal development process. A University of Hawaii case explored some of the benefits and criticisms of PSP [12]. The study demonstrated that students using PSP developed a stronger appreciation for utilizing a structured process. However, the study also noted PSP's "crushing clerical overhead," which could result in significant data recording errors.

Interestingly, Humphrey's Team Software ProcessSM (TSPSM) [13] corroborates the fusing of individual estimation efforts into a team-level estimate. Individual members of TSP teams practice PSP and employ individual PSP-gathered measures in making team-level estimates.

Researchers at Auburn University have developed a lightweight alternative to PSP, known as Principle-Centered Software Engineering (PCSE) [14]. PCSE uses a proxy-based approach to estimating SLOC, which is clearly lighter weight, yet still relies on non-trivial, mathematical models to produce a result. On one hand, the lightweight nature of PCSE calculations overcomes the heavy mathematical models of PROBE. On the other hand, the use of SLOC limits the usefulness of PCSE in graphical or web-based development, which may include graphics, user interface widgets, etc.

The informal survey of ASP members reveals anecdotal evidence that software engineers typically opt for a "gut-feel" approach, where effort estimation is based on no more than educated guesses [15]. This impression is especially true of software engineers working outside the constraints of an organization, which typically mandates the use of formal tools and processes. Since developers typically estimate 20-30% lower than actual effort [16], this would explain why the gut feel approach is typically heavily padded.

A "Better Than Guessing" Agile Approach

A void exists in the spectrum of effort estimating tools, specifically focused on lightweight tools for the individual. Sort Insert Size Estimate (SISE) represents a new approach to forecasting future software development effort by combining a standard regression model with relative task sizing. SISE introduces an agile approach to sizing by the individual in much the same way Planning Poker introduced agile sizing to the team.

Specifically, the SISE model guides the estimator through the process of organizing future tasks, not by matching them to historical analogies, but by size ranking, relative to historical tasks with known effort measurements. The SISE model then derives its results based on a simple principle: if the perceived size of a future task falls in between the actual size of two historical tasks, then the actual effort of the future task should also lie between the actual effort of the two historical tasks.

For example, assume two tasks have been completed on a project by a software developer: the first task in four hours and the second in six hours. A third task is then assigned to the developer, who estimates the relative size of the task to be somewhere between the first two tasks. It can be assumed, then, that the actual effort for the third task is somewhere between four and six hours. Note that this approach does not necessarily produce a single value estimate, but rather an upper and lower bound for the estimate (i.e. prediction interval).

If a single-value estimate is required, it can be extracted from the prediction interval in a number of ways. One approach to deriving a specific value for the estimate is to take the upper bound values. This produces a high confidence estimate, yet strongly resembles the practice of "padding" the estimate (i.e. playing it safe). Another approach involves simply averaging the upper and lower values and relying on the law of averages to even out the errors. In many instances, this approach may produce a specific value that is "good enough" for the circumstances with a minimal amount of effort and complexity. A third approach relies on a simple statistical analysis of the developer's historical data to produce a weighting factor which is applied to the upper and lower bounds to produce a single value. Simply put, the weighting factor represents where, on average, the developer's actual effort for all tasks fell between the upper and lower bounds of the estimates for those tasks. For example, a developer who, on average, completes tasks exactly at the midpoint between the upper and lower bounds of the estimate will possess a weighing factor of 0.5.

In its simplest form, the SISE model is specifically targeted at individual software developers, such as microISV operators, independent consultants, and team members. The approach's strength lies in the fact that individuals may develop reasonably accurate personal estimates based on their own historical data, while excluding team-level factors such as communication and overhead costs. Although the factors involved in a team-level effort estimate are more numerous, project-level estimates may also be derived with the assistance of the SISE model by combining individual team members' estimates and applying existing, proven approaches to adjusting for overhead. Another, more radical application of SISE might involve treating entire projects as tasks and deriving an estimate in the same manner as used by individual developers. This type of application would be most useful in certain organizations, which insist on estimates extremely early in the development cycle (i.e. before requirements are fully elaborated and understood).

Obviously, the SISE approach requires a calibration period, during which a historical data pool is constructed for deriving future estimates. Fortunately, many organizations and individuals already track effort expenditures, via time sheets or project management tools, providing a ready source for historical data. This leaves a historical gap for new developers and for environments lacking historical records. Fortunately, the calibration period can be relatively short, depending on the typical size variations in tasks; organizations with task sizes that vary widely will require a longer calibration period to derive a suitable data pool of historical values. Although not recommended, in the absence of historical data, it is possible to use another software engineer's actuals as a surrogate data pool and then rotate surrogate data out as actual developer data becomes available.

A variety of other factors exist which may affect the accuracy or precision of an estimate. These factors—such as programming environment changes, statistical outliers in the data set, data recording and accuracy, and granularity—must be addressed in a consistent manner when implementing the SISE model in an organization's environment. The key to successfully applying this approach lies in the individual's ability to recognize and adjust for these factors.

Conclusion

The emergence of software written by one-person teams—mobile device apps, services, components—renders inaccurate the portrayal of software engineering exclusively as a team activity. It brings a vanguard of exciting concepts in which the individual plays a pivotal role in not only creating viable software, but in controlling the development process. Systematic effort estimation, once the province of teams, has benefit to individuals as well; however, it, like so many other software methods, has to be stripped of unnecessary tasks if individual developers are to reap that benefit. It has to be intuitive, usable, and produce results that are more accurate than outright guessing.

Further Information

The SISE model is currently under development at Auburn University by IT veteran and graduate student Russell Thackston. The model is currently under evaluation by Auburn University's Computer Science and Software Engineering program.

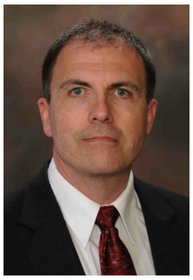
Disclaimer:

PSPSM and TSPSM are service marks of Carnegie Mellon University.

ABOUT THE AUTHORS

Russell Thackston, MSCIS, is a graduate student in Auburn University's Department of Computer Science and Software Engineering. He has spent more than 15 years developing business solutions as a software developer and consultant in the retail, commercial, and open source environments. He served in the U.S. Air Force during Desert Storm and Desert Calm and has been happily married for 25 years.

Phone: 334-246-2600
Fax: 334-844-6329
E-mail: Russell.Thackston@Auburn.edu



David A. Umphress, Ph.D., is an associate professor of computer science and software engineering at Auburn University, where he specializes in software development processes. He has worked over the past 30 years in various software and system engineering capacities in military, industry, and academia settings. Umphress is an IEEE certified software development professional.

Phone: 334-844-6335
Fax: 334-844-6329
E-mail: David.Umphress@Auburn.edu

REFERENCES

1. "July 2009 Metrics Report." AdMob Metrics. Web. 14 Aug. 2011. <http://metrics.admob.com/2009/08/july-2009-metrics-report/>.
2. "Apple Developer Programs 2011." Apple Developer. Web. 14 Aug. 2011. <http://developer.apple.com/programs/>.
3. "Android Market Developer Signup." Android Market. Web. 14 Aug. 2011. <https://market.android.com/publish/signup/>.
4. "Gartner Says Worldwide Software as a Service Revenue Is Forecast to Grow 21 Percent in 2011." Technology Research | Gartner Inc. Web. 14 Aug. 2011. <http://www.gartner.com/it/page.jsp?id=1739214>.
5. "ASP Member Forum." Association of Software Professionals. Web. 14 Aug. 2011. <http://members.asp-software.org/newsgroups/showthread.php?t=25806>.
6. Cohen, Mike. "Succeeding with Agile." Reading, MA: Addison-Wesley. 2010. Print
7. Jorgensen, Magne, and Martin Shepperd. "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions on Software Engineering 33.1 (2007): 33-53. Print.
8. Boehm, Barry W. "Software Engineering Economics." IEEE Transactions on Software Engineering SE-10.1 (1984): 4-21. Print.
9. "Standish Chaos Report 2009," available at <https://secure.standishgroup.com/reports/reports.php>
10. Jorgensen, Magne, and Martin Shepperd. "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions on Software Engineering 33.1 (2007): 33-53. Print.
11. Humphrey, Watts S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995. Print.
12. Philip M. Johnson and Anne M. Disney, "The Personal Software Process: A Cautionary Case Study," IEEE Software, November/December 1998: 85.
13. Humphrey, Watts S. "Introduction to the Team Software Process". Reading, MA: Addison-Wesley. 2000. Print.
14. "Principle-Centered Software Engineering" Auburn University Web. 12 Oct. 2011. <http://swemac.cse.eng.auburn.edu/~umphrda>.
15. "ASP Member Forum." Association of Software Professionals. Web. 14 Aug. 2011. <http://members.asp-software.org/newsgroups/showthread.php?t=25806>.
16. Michiel Van Genuchten, "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development," IEEE Transactions on Software Engineering, Vol. 17, No. 6 (June 1991): 582. Print.

Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications, is seeking dynamic individuals to fill several positions in the areas of software assurance, information technology, network engineering, telecommunications, electrical engineering, program management and analysis, budget and finance, research and development, and public affairs.

To learn more about the DHS Office of Cybersecurity and Communications and to find out how to apply for a vacant position, please go to USAJOBS at www.usajobs.gov or visit us at www.DHS.GOV; follow the link Find Career Opportunities, and then select Cybersecurity under Featured Mission Areas.

Applying Scrum to Stabilize Systems Engineering Execution

Richard Carlson, Boeing
Philip J. Matuzic, Boeing
Robert L. Simons, Boeing

Abstract. Years before the Agile Manifesto [1] was created in 2001, effective practices for managing projects were being applied to reduce project cycle times and reduce costs while increasing productivity and quality. Agile [2] principles that emerged from the manifesto promulgated rapidly throughout industry on software development projects at first, and eventually into projects that were not software centric. This article focuses on the application of Scrum [3], an agile project management method that uses simple practices to enable stability for the execution of systems engineering (SE) activities and the development of requisite systems engineering work products throughout the product development lifecycle.

Introduction

The background and history of using agile on software development projects are well documented by the IEEE, the National Defense Industry Association conferences, and through workshops, papers and technical articles published by the SEI. The successes of agile software development projects have attracted tremendous interest throughout the SE community. There has been an upsurge within academia and industry advocating the use of these practices and principles to implement agile SE and reports of successes are emerging. This article identifies key drivers that are paramount in stabilizing agile-based SE product development efforts, lessons learned from empirical experience, specific areas of practice that were difficult to implement, and requisite infrastructures that must be in place before considering agile at the SE level.

Agile Systems Engineering Stability Drivers

There are some interesting contrasts between traditional (or waterfall [4]) project management and agile project management:

- High overhead vs. focused work: Traditional projects suffer excessive overhead caused by lengthy meetings that waste valuable team time. Agile projects concentrate on guiding committed work in progress through short, time-boxed ceremonies.
- Mature designs vs. incremental development: Traditional projects place too much emphasis on first completing mature architecture and design including features and functions that may rarely, if ever, be used. Agile thinking demands that the product and its associated artifacts be developed incrementally through a series of short iterations guided by frequent customer feedback throughout product development.
- Command-and-control vs. empowered teams: Project managers are trained (and certified through formal project management training) to exercise authority and direction of project members in order to accomplish predefined and measured goals. Agile projects, with participation of the existing project management authority, allow teams to self-manage and self-organize so they can become increasingly synergistic and efficient.

Agile Project Planning

The goals of agile project planning are defined in four critical artifacts: product vision, initial backlog of requirements, roadmap, and release plan.

The product (or project) vision must be created by the project visionary, typically the product owner. When implementing a Scrum approach, the product owner coordinates with key stakeholders to describe a desired state of the product in a series of release increments. The vision should be a clear and succinct statement describing the product's capabilities and features that set it apart from anything built in the past. The vision statement must convey the expected value expressed by the customer or as specified in the contract. It should also include the company's or project's motivation of using Scrum to manage its development, the product's key features, potential, and most significant risks with corresponding mitigation strategies.

The backlog, or product backlog as it is coined in Scrum, is a prioritized list of tasks encompassing everything essential to the success of the product. The priority of items listed in the backlog is typically driven by the customer.

The product roadmap is a translation of the product vision into product feature development terms. It describes "what" needs to be completed or implemented, and it should map development timelines of the product that reflect major features, a set number of releases, both internal and external, and the number of iterations planned for each release cycle. The roadmap should be updated prior to the start of each release cycle to ensure changes directed from the customer or as a result of product backlog re-prioritization are communicated to the core project team and its key stakeholders.

Release planning involves key stakeholders in determining incremental releases for the product development. The release plan describes the "how" and includes the details of the product roadmap. It also includes details of product development that extend throughout the lifecycle of the project.

About Scrum

Scrum is an agile framework for managing complex projects. Originally Scrum was formalized for software development projects, but works well for any complex, innovative scope of work. The possibilities are endless.

The Scrum framework is deceptively simple:

- The product owner creates a prioritized feature wish list called a product backlog.
- The team has a defined amount of time, called a sprint, to complete its work. A sprint is usually two to four weeks and the team meets each day to assess its progress (called the daily Scrum).
- During a sprint [5] or iteration planning, the team pulls a small quantity of tasks from the top of the wish list—the sprint backlog—and decides how to implement those features during the current sprint.
- During the sprint, the Scrum master keeps the team focused on its goal.
- Task results at the end of the sprint are an executable increment of the product which is potentially deployable to a customer, put on a storage shelf, or demonstrable to a stakeholder.
- The sprint formally ends with a sprint review and team retrospective.
- The next sprint begins with sprint planning where the team chooses another chunk of the product backlog and begins working on the new features.

The cycle repeats until enough items in the product backlog have been completed to finalize the product, the budget is depleted, or a programmatic deadline arrives. Which milestone marks the end of work is entirely specific to each project. No matter which impetus stops work, Scrum ensures completion of the most valuable work before the project ends.

Scrum employs an iterative and incremental method for managing projects. Scrum has no ties to the Project Management Institute [6] and it is not part of the Project Management Body of Knowledge [7]. Although Scrum was originally recommended for software development projects, it is easily applied to just about any type of project. This means that Scrum can and has been used as a framework to manage a wide range of non-software-centric projects.

Scrum uses a set of simple practices that drive development stabilization:

- Scrum relies on active customer participation. Open communication with the customer provides visibility into issues and problems that may have adverse effects on project schedule, cost, and product release. When the customer is multitasking or otherwise not available to participate in technical interchanges, such as product reviews, they invite problems that will cause things to go awry.
- Daily stand-up meetings are short, time-boxed meetings that keep the team focused on communicating what they have accomplished since the last meeting with other team members, what they plan to do next, and any impediments that are keeping them from achieving planned work.
- Extensive planning is conducted prior to iteration activities to ensure development environments are available and that team members have everything they need to complete their tasks. Planning is conducted on the first day of each iteration so the team knows the overall goal and can determine how all work will be completed. Iteration planning assures the team commits to an amount of work it can complete during the iteration with a high probability of success.
- Estimating work in terms of requirements is conducted by all key stakeholders at the beginning of every project, and by the team at the start of each iteration throughout the project.
- Iterative development involves short, single passes through an entire development lifecycle. Short iterations are necessary to obtain early and frequent feedback from customers and other key stakeholders.
- Prioritized work contained in a product backlog encompasses all project requirements. Priorities are based on customer decisions or are business driven to ensure that only requirements bringing the most value to the customer are completed first.
- Reviews of work products completed during the iteration assure that adequate verification and validation take place.
- Self-organized agile teams must be empowered to make decisions, consult with domain and subject-matter experts, and select work tasks in an appropriate and logical sequence.

The Scrum Process

Scrum is a simple framework with three roles, three critical artifacts, and four low-overhead ceremonies (or meetings). The roles are:

Product Owner: The product owner represents the voice of the customer and is accountable to ensure that the team delivers value to the business. The product owner is responsible for taking all the inputs defining the product from the customer or end-user of the product, as well as from team members and stakeholders, and translating them into a product vision. In some cases, the product owner and the customer are one and the same; in other cases, the customer might actually be millions of different people with a variety of needs. The product owner writes customer-centric items, prioritizes them, and adds them to the product backlog. Scrum teams should have one product owner and, while they may also be a member of the development team, it is recommended that this role not be combined with that of the Scrum master. [Note: There are many instances of multiple individuals filling the role of the product owner to assure domain or technical knowledge is available to the team.] The product owner role maps to the product (line) manager position in many organizations.

Scrum Master: Scrum activities are facilitated by a Scrum master, also written as ScrumMaster, who is accountable for removing impediments enabling the team to deliver the iteration or sprint goals and deliverables. The Scrum master is not the team leader but acts as a buffer between the team and any distracting influences. The Scrum master ensures that the Scrum process is followed and is the enforcer of rules. A key part of the Scrum master's role is to protect the team from distraction and keep them focused on the tasks at hand. The role has also been referred to as servant-leader to reinforce these dual perspectives. The Scrum master is responsible for helping the team be successful. The Scrum master is not the manager of the team; he or she serves the team helping remove impediments, facilitating meetings, and supporting the practice of Scrum.

Team: The team is responsible for developing and delivering the product and is typically made up of five to nine people with cross-functional skills. The team should be self-organizing and self-managed. SE teams using Scrum should be staffed with functional and technical SEs, a domain-knowledgeable architect and software engineer, and testers.

Scrum Artifacts and Transparency Tools

The product backlog is a prioritized list maintained throughout the entire project. It aggregates backlog items that are broad descriptions of all potential features prioritized by business value as an absolute ordering. The product backlog is "what" will be built, sorted by importance. It is open and editable by anyone and typically contains rough estimates of business value and/or development complexity. Those estimates help the product owner gauge the timeline and, to a limited extent, prioritize tasks.

The iteration backlog is the list of work the team must address during the next iteration. Selected product backlog items are broken down into tasks, which, as a best practice, should normally be between four and 16 hours of work. With this level of detail, the whole team understands exactly what to do. Tasks on the iteration backlog are never assigned. Instead, tasks are selected by the team members as needed, according to the set priority and team member skills. This promotes self-organization of the team. The iteration backlog is the property of the team, and all included estimates are provided by the team. However, the Scrum master may prefer to maintain this artifact to ensure it reflects iteration status in real-time. Often an accompanying taskboard or work-in-progress (WIP) board is used to view and change the state of the tasks during the current iteration.

The taskboard is a transparency tool that shows tasks in work during an iteration. The taskboard is updated by team members as they complete each task. As the tasks are accepted by the team members, they are checked out. Tasks are worked through completion and must be verified by the product owner before initiating another task. The team maintains the taskboard up to the "To Be Verified" column. Verification is done by the product owner and shown by the product owner moving the task to the completed column. If a team member completes a task, he/she cannot take credit until the product owner verifies it is complete. This is one of the most simple yet best transparency tools used on agile projects!

The iteration burndown chart displays a metric of remaining work in the iteration backlog. Updated every day, it provides a view of ongoing iteration progress.

Scrum Ceremonies

The daily Scrum or stand-up meeting is conducted each day and is facilitated by the Scrum master. During the meeting, each team member responds to three questions:

- What have you done since the last stand-up meeting?
- What are you planning to do today?
- Do you have any impediments that would prevent you from accomplishing your work?

The Scrum master facilitates resolution of these impediments, although all resolution occurs outside the daily Scrum itself to keep the meeting under 15 minutes. This meeting must always start on time, be conducted at the same location and time every day, and while anyone may attend, only team members are allowed to speak. Management is not allowed to speak and topics outside the strict agenda are scheduled for separate discussion.

Iteration planning for execution is conducted on the first day of the iteration. The core Scrum team (i.e. product owner, team, and Scrum master) meet to determine features that must be completed during the next iteration. At the opening of the meeting, the product owner explains the vision and product roadmap. From the product backlog of Prioritized Backlog Items (PBIs) he/she identifies "what" needs to be completed next. This meeting, facilitated by the Scrum master, consists of detailed planning activities between the product owner and the team. PBIs, identified by the product owner for the next iteration are estimated for effort and complexity and then selected by the team. The number of PBIs selected is dependent on the team's capability or their availability to complete selected PBIs during a single iteration. If the selected PBIs can be completed with a high level of confidence during the upcoming iteration, the team commits to those PBIs and the Scrum master enters the items into the iteration backlog. If they cannot be completed due to over-complexity or size, they are either decomposed or deferred unless they are the next highest priority items. During the second half of the iteration planning meeting, the team determines "how" to complete the selected PBIs by breaking down each item into quantifiable tasks and activities. This promotes "systems thinking" and ensures that all requisite and value-added steps are identified and implemented. As a standing rule for Scrum teams, all tasks must be completed and verified before a PBI can be declared "done" or available for deployment or delivery.

At the end of the iteration, team members conduct a review of all completed work with key stakeholders to validate the fact that requirements were interpreted accurately. The review sets up a conversation between the participants about what was done, a demonstration of prototypes completed, and a decision on what to complete next. Required iteration review attendees include all team members, the product owner, Scrum master (who facilitates the review), relevant key stakeholders, customer or customer representatives, users, and any interested engineers, domain experts, and managers. Attendance by all is vital to ensure that everything completed is presented, questions are properly responded to, and that feedback is given and received. During the review, the project is assessed against the iteration goal established during the iteration planning meeting. The Scrum master ensures the review does not exceed its time-boxed schedule (usually four hours or less). The team should be prepared to discuss what was done, how the iteration's goals were met, and to demonstrate the product in its current state. At the conclusion of the review, the product owner acknowledges either acceptance of work products presented or deferral of work until it is more mature or good enough for release into product development or production.

Retrospectives are conducted to build team commitment, transfer knowledge to the next iteration, and share information with the customer, management, and other teams. The retrospective is the structured reflective practice that enables teams to learn and improve based on empirical experiences. It is a time-boxed meeting held with the team members, product owner, and Scrum master at the end of an iteration to:

- Discuss what was successful about the iteration or release.
- Realize what could be improved.
- Learn from experiences and plan for subsequent iterations.
- Plan to incorporate the successes and improvements in future projects.

- Share and pass along the learning experience.
- Make changes for the next iteration.

Retrospectives are conducted at the end of the iteration; however, a retrospective is very effective when conducted at the end of any event, that is, any time there is value for the team to pause for a few minutes to learn from its recent experiences. But be very careful; retrospectives are not conducted to identify mistakes and place personal blame or personal attacks. They should not involve the resolution of issues and problems, and should never become a planning meeting.

Agile Helps to Stabilize Systems Engineering Defects

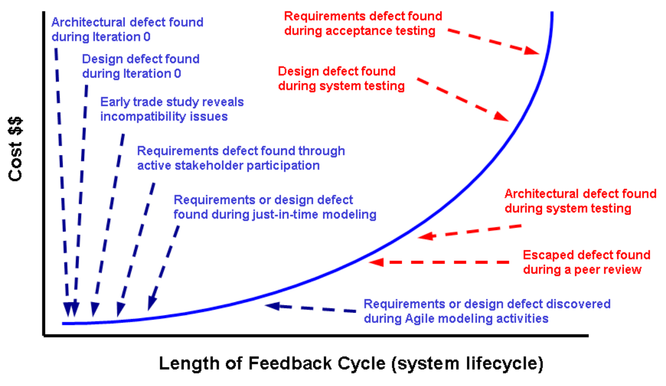
Mapping the potential costs of defects found by various detection techniques to common development strategies versus the cost-of-change curve very clearly shows that using an agile approach is less costly than traditional approaches. It has long been realized that the earlier in the lifecycle a defect is corrected, the less costly. In the chart below, errors detected early and often by self-organizing teams who develop work products iteratively are much cheaper to fix than latent defects detected later in the project lifecycle. [Source: Numerous articles and papers published on the subject.]

Agile/Scrum Practices for SE

Agile SE for software development was first implemented in Boeing on two very large programs consisting of multiple engineering domains. Agile SE teams defined and developed requirements and functions in the form of user stories [8], developed and demonstrated prototypes and received real-time user feedback, and created a product backlog for software development.

Using agile to conduct SE activities and create SE work products is a viable approach to reduce overhead and lengthy schedules. Most of the agile principles and Scrum practices can be applied to SE teams, for example:

- Conducting four-week “requirements iterations”: Considering the amount of coordination and collaboration with domain and subject-matter experts required, four-week iterations is a good point of departure for requirements iterations. During project execution, if the team feels they can improve productivity by changing the iteration duration, then it should be allowed. Test for a month or two to validate the increase in productivity is working.
- Staffing systems engineering teams with systems engineers, business and/or functional analysts, tester, and at least one domain-knowledgeable software engineer. Much of the verification work is accomplished using this mix of expertise.
- Empowering the team by providing them with the requisite authority and everything they need to be fully functional and productive.
- Developing SE work products incrementally using the agile principles and Scrum practices.
- Writing requirements in user story format including requisite acceptance tests.
- Conducting iteration planning meetings on the first day of every iteration.
- Conducting brief daily stand-up meetings (15 minutes or less).



Inspired by Scott Ambler (http://www.ambysoft.com/essays/agileTesting.html)

Figure 1 – Defect Cost Varies By Feedback Cycle

- Conducting peer reviews for all significant work products.
- Ensuring systems and sub-systems integrated verification and validation.
- Holding iteration reviews of SE work products including prototypes, documentation, trade studies, analyses, user stories, and groomed product backlogs.
- Conducting iteration retrospectives at the end of each iteration, and encouraging holding frequent mini-retrospectives.
- Holding Scrum of Scrums meetings with Scrum master and other stakeholders at least two or three times a week.
- Establishing an active product owner core team consisting of multiple product owners on multiple domain projects, and projects that require multiple product owners to provide sufficient domain and subject-matter expertise.
- Supporting geographically distributed teams with simple transparency tools to enable a highly collaborative and visible working environment.
- Implementing agile on large programs across multiple engineering domains.

Key Things Learned Using Agile/Scrum

The following are examples of what was learned implementing agile on SE projects:

- Scrum is a natural approach to managing a project, and teams like the simplicity of the Scrum framework.
- Close collaborations improve significantly between team members and key stakeholders using Scrum.
- Impediments, issues, problems, and potential risks are identified early and often in real-time through short, daily stand-up meetings.
- Retrospectives can become boring and mundane if action plans are not implemented aggressively and followed through.
- Periodic Scrum of Scrums meetings on large programs optimize communications and information sharing across project teams and the greater organization.
- Product owners with sufficient domain knowledge and overall understanding of the product are hard to find and retain.
- Scrum masters are daunted when attempting to facilitate more than two teams.
- Adoption of agile on SE projects is slow due to an overall resistance to change.
- Simple inexpensive transparency tools are very effective (e.g. WIP board, backlogs, burndown charts).

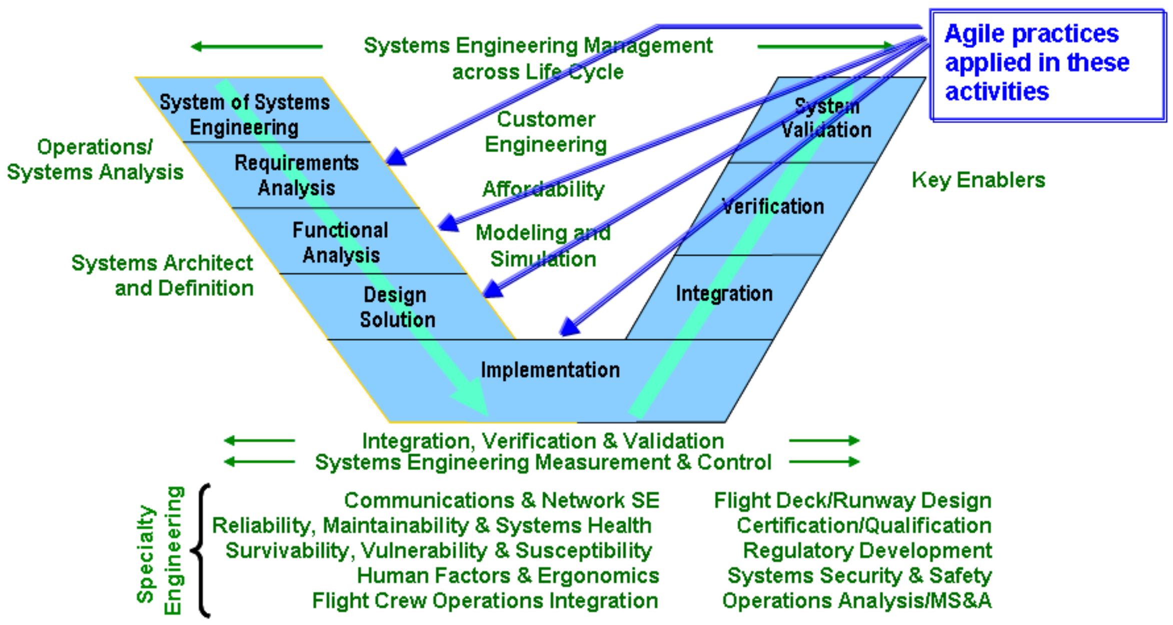

Systems Engineering Management across Life Cycle
Agile practices applied in these activities
System of Systems Engineering
Requirements Analysis
Functional Analysis
Design Solution
Implementation
Integration
Verification
System Validation
Operations/ Systems Analysis
Systems Architect and Definition
Customer Engineering
Affordability
Modeling and Simulation
Key Enablers
Integration, Verification & Validation
Systems Engineering Measurement & Control
Specialty Engineering
Communications & Network SE
Reliability, Maintainability & Systems Health
Survivability, Vulnerability & Susceptibility
Human Factors & Ergonomics
Flight Crew Operations Integration
Flight Deck/Runway Design
Certification/Qualification
Regulatory Development
Systems Security & Safety
Operations Analysis/MS&A

Figure 2 – SE V Diagram With Agile

The Systems Engineering "V" and Agile Enablers

An effective way to see where agile can be applied to SE is to revisit the SE V-model (see below). The fundamental SE processes include:

- Requirements analysis
- Functional analysis and allocation
- Synthesis and integration
- Verification and validation
- Systems analysis and control

The left side of the "V" involves identification and decomposition of requirements, functions, and design. The right side of the "V" includes the integration and characterization of product development. All of the attributes supporting the fundamental SE processes either have been or can be enabled using Scrum's simple practices.

During early system development an initial examination of customer needs, often referred to as a statement of need, is always necessary, but is especially critical if initial customer requirements are not available. The first product generated from examination of the statement of need is a feasibility analysis. The application of Scrum at this early stage introduces the framework for developing follow-on functional system requirements.

Alternative technology options and design considerations can quickly be identified and assessed during product backlog development activities. Because the customer's view is represented through the product owner, the team has this perspective available to them as they quickly identify, assess, and document their findings.

The strength of the Scrum process of infusing customer insights through the product owner throughout the planned series of iterations performed is important. Quality Function Deployment (QFD) [9] is often used in systems engineering customer engagement activities. The QFD is designed to establish a communications dialog and elicit customer input on end-state goals and objectives. The findings of a QFD are used to frame the contents of the statement of need and reflect the customer's perspective in the feasibility analysis report. A fundamental weakness of QFD is that it is almost exclusively performed as a single activity, usually early in the feasibility analysis timeline. Because this is an instant-in-time view of the customer's perspective, emerging customer concerns and needs are lost. The Scrum framework of continual customer participation throughout an incremental lifecycle provides the most current customer view for the system development process. Applying a disciplined Scrum approach ensures the feasibility analysis has customer buy-in throughout the study. The resultant Scrum-based feasibility analysis establishes a customer-aligned baseline that drives the development of operational user requirements, conceptual designs, and follow-on system-level requirements and specifications.

Conclusion

The Scrum agile approach is by far the most widely used and implemented technique [10] adopted by software and systems development teams because of its simple practices and empirical process control [11]; but Scrum is a significant cultural shift from more traditional ways of project management. Scrum roles differ significantly from traditional roles and frequently cause confusion among general engineering and business communities alike. Scrum terminology is foreign to most and change from the status quo is not only difficult, but is often initially resisted. Scrum is most effective when used as a wrapper for the organization's existing engineering practices and improved as necessary during product delivery or deployment increments. Scrum provides an environment where everyone can feel good about their job, their contributions, and that they have done their very best.

ABOUT THE AUTHORS

Dick Carlson has worked for Boeing for more than 7 years. He retired from the U.S. Army where he spent the bulk of his career in communications-electronics and systems engineering. After the Army, Dick worked as a consultant and employee focused on the development of software systems and engineering technologies supporting DoD, commercial, and private industry.

He has been active for nearly 15 years in the implementation of agile practices on a variety of software development and non-software centric projects. Dick spends most of his work time coaching, mentoring, and training Boeing engineers on the application of agile implementation and deployment. Dick has a Bachelor of Science degree from the University of Maryland, and is recognized by the Scrum Alliance as a Certified Scrum Professional and Certified Scrum Master, and is certified in Lean-Agile Project Management.

E-mail: richard.carlson2@boeing.com
Phone: 714-350-9946

Philip J. Matuzic is an Associate Technical Fellow at the Boeing Company Satellite Development Center, in El Segundo, California, where he is Chief Software Technologist and lean-agile modeling proponent. Philip also consults and provides technical training for Boeing, IBM, Northrop Grumman, Raytheon, Lockheed, and the U.S. Government. Mr. Matuzic has a MS in Software Management from Carnegie Mellon University, and a Project Management Certificate from the California Institute of Technology.

E-mail: philip.j.matuzic@boeing.com
Phone: 310-364-7387

Robert (Rob) Simons is a Boeing Technical Fellow in Boeing's Defense System's System Engineering organization in St. Louis. Rob is responsible for systems engineering and analysis supporting C4I, Network Centric, and homeland security activities. He holds multiple roles in program, project and team initiatives, and leads several academic collaboration initiatives. Rob holds MS degrees in Telecommunications and Engineering Management and an Graduate Certificate in Project Management from Washington University in St. Louis, an MBA from Lindenwood University and an MA in International Relations from Webster University.

E-mail: robert.l.simons@boeing.com
Phone: 314-234-3107

REFERENCES

1. Agile Manifesto <www.agilealliance.org/the-alliance/the-agile-manifesto>
2. Agile Alliance <www.agilealliance.org>
3. Scrum Alliance <www.Scrumalliance.org/learn_about_Scrum>
4. Waterfall model <http://en.wikipedia/Waterfall_model>
5. Sprint <http://en.wikipedia.org/wiki/Sprint_(Scrum)>
6. Project Management Institute <www.pmi.org>
7. Project Management Body of Knowledge <www.pmi.org/PMBOK-Guide-and-Standards.aspx>
8. User stories <http://en.wikipedia.org/wiki/User_story>
9. What is QFD? <http://www.qfdi.org/what_is_qfd/what_is_qfd.htm>
10. Figure 1 - Agile Is Primary Approach <http://www.practiceagile.com/>
11. What's Unique About Scrum? <http://scrummethodology.com/>

Architecting for Sustainable Software Delivery

Ronald J. Koontz, Boeing
Robert L. Nord, SEI

Abstract. With increasing emphasis on avionics system rapid development and reduced cycle times, software architecting practices can be applied with agility to enhance evolving stakeholder concerns while sustaining long-term business goals.

Software intensive systems, and in particular military avionics platforms, are facing both shrinking defense budgets and the continued expectation for more advanced mission capabilities. The business case is that it is much more affordable to extend existing platform capabilities than to consider new platform designs. Over a product lifecycle, business goals and objectives continue to evolve as capabilities are realized. Paramount to enhancing current platform capabilities is an extensible and sound avionics system and mission-computing software architecture.

This article describes the role that five architecture practices are continuing to play in enabling the Apache Block III program to achieve long-term business goals. Agility is applied according to Jim Highsmith's definition, which describes it in terms of balancing flexibility and stability [1]. Such agility enables architectural development to follow a "just-in-time" model that complements iterative and incremental enhancement development and integration [2]. Delivery of capabilities is not delayed pending the completion of exhaustive requirements and design activities and reviews. At the same time, architectural agility maintains a steady and consistent focus on continual architectural evolution in support of emerging capabilities.

The Networked Common Operating Real-Time Environment (NCORE) software architecture for the Apache Longbow helicopter Mission Processor provides a case study to illustrate the architecting practices that support agility and sustainment of long-term business goals. The NCORE architecture was initially developed and flight-tested during the jointly funded Army Aviation Technology Directorate (AATD) and Boeing Manned/Unmanned Common Architecture Program, Phase II (MCAP II). The MCAP II risk reduction program initially focused on the evaluation of emerging software technologies such as real-time Java.

Since then, the NCORE architecture continues to evolve on the Apache Longbow Block III Program. The driving architectural requirements include safety, performance, availability/reliability, modifiability, and interoperability. As initial platform capabilities are realized and as avionics computing lifecycles shorten, increased emphasis is placed on extensibility and the desire to host applications developed by third parties. In parallel, embedded infrastructure software must be architected to reduce overall time and cost to incorporate hardware upgrades (proactive obsolescence management) and to enable the hosting of new or existing application-level software.

In the section that follows, each practice is described and illustrated with an NCORE architecture example. Next, incremental delivery of new capabilities is described in terms of how it is realized by combining all of the practices. Finally, the essential characteristics of the practices are summarized according to agile development and architecture criteria. This summary provides a checklist to aid learning for others developing software-reliant systems, and provides feedback on whether their application is on track to help meet project goals.

Architecture Practices That Balance Flexibility and Stability

Architecture best practices are a set of actions, methods, techniques, and/or strategies applied to software architecting and the software lifecycle that are well proven and known to yield desired outcomes without introducing unnecessary program risk. Those architecting practices that are leveraged by the NCORE architecture and that can be broadly applied to avionics platforms are now described and analyzed from the point of view of agility.

Incremental/Iterative Development

NCORE architecture and application software artifacts are developed using an incremental and iterative development lifecycle [3], notionally shown on a fiscal year calendar in Figure 1. Based on periodic customer statements of work, incremental capabilities are planned, developed, tested, and fielded. For each statement of work iteration, integrated build and release plans are developed. To enhance testability and integrability, software builds contain two or more mini-builds that accelerate design, development, automated testing, and platform integration.

This incremental/iterative development approach parallels agile software development, with cross-functional/agile teams of requirements/integrators, coders, and testers working according to agreed-upon release planning (the build plan). Incremental/iterative development further enables user feedback and refinement, fixes, and overall product enhancements to be folded back into product deliverables as the software matures.

Iterative loops within each build cycle map to mini-builds, and the quantity of mini-builds is adjusted to accommodate the functional complexity of the build. For example, a typical six-month build cycle may be decomposed into two or more mini-builds where multiple parallel teams execute independent work threads. Each team defines incremental build content that can most effectively produce overall build content objectives at the time of final build release. Incremental build content definition minimally considers work thread complexity and resource availability relative to overall build objectives.

Informed Technology-Insertion Decision Making

Informed technology-insertion decision making is built upon communication and knowledge sharing and is characterized as a cyclic and iterative process of understanding stakeholders' concerns, making and documenting decisions, and evaluating the consequences. This communication provides near real-time, two-way dialog between architects and stakeholders. Both push and pull communication strategies are concurrently employed:

Push: architecture and software design documentation. Information about the architecture is periodically provided to stakeholders.

Pull: architecture evaluations. Periodic architecture evaluations collaboratively pull information from the business management, architecture team, and stakeholder community.

Together, these activities contribute to enhanced knowledge sharing across the integrated team.

The NCORE architecture description is centered on module, Component-and-Connector (C&C), and allocation views [3]. Several styles employ separation of concerns to capture architecturally significant artifacts. To maximize resources and avoid duplication, overlapping stakeholder concerns are combined by the architects into a concise set of architectural views. As the architecture evolves, the architects carefully analyze current concerns and anticipate future stakeholder needs to determine whether new views are required or whether existing views can be enhanced.

Table 1 shows the stakeholder-to-documentation-artifact-type matrix developed by the architecture team when initially developing NCORE architectural views. According to the Table 1 key, an individual stakeholder level of concern is identified as either "detailed," "some details," "overview," or "anything." The "anything" concern level signifies access to all readily available artifacts which can be browsed by any new stakeholder seeking rapid and broad architecture understanding. Software architects and development team members seek "detailed" Module and C&C view content, which convey static and runtime concerns, respectively. Overall, decomposition and layered module views are most popular across the stakeholder community based on the multi-faceted and layered NCORE architecture and the overlapping concerns they address.

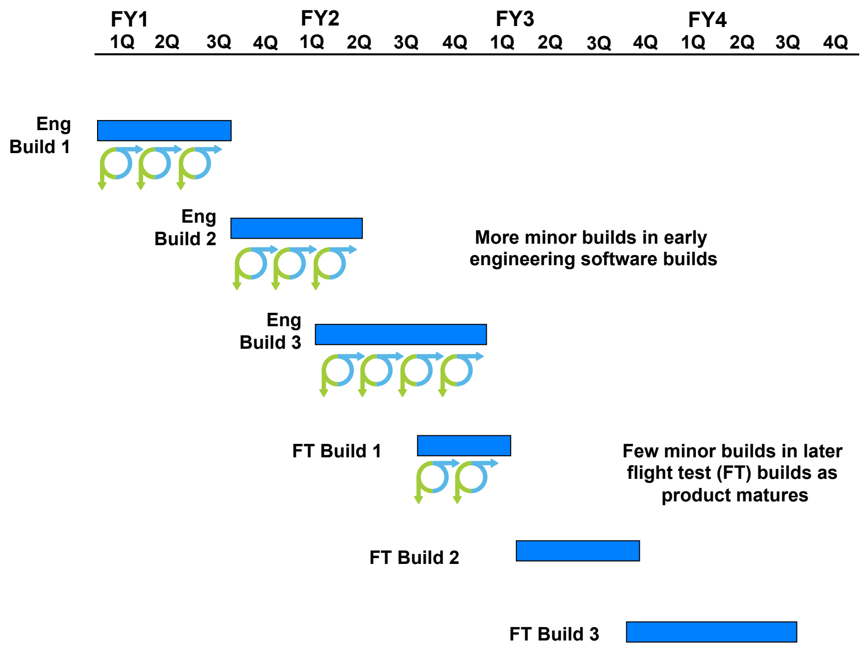

FY1
FY2
FY3
FY4
1Q 2Q 3Q 4Q 1Q 2Q 3Q 4Q 1Q 2Q 3Q 4Q 1Q 2Q 3Q 4Q
Eng Build 1
Eng Build 2
Eng Build 3
FT Build 1
FT Build 2
FT Build 3
More minor builds in early engineering software builds
Few minor builds in later flight test (FT) builds as product matures

Figure 1: Notional NCORE Incremental/Iterative Development Lifecycle

	Module Views				C&C Views				Allocation Views			Other
Stakeholder	Decomposition	Generalization	Uses	Layered	Publish-Subscribe	Shared-Data	Client-Server / Peer-to-Peer	Communicating-Processes	Deployment	Implementation	Work Assignment	SOW & Performance Specifications
Current and future architects	d	d	d	d	d	d	d	d	d	s	o	d
Government project management	d	o	s	o	s	s	o	o	s	o		d
Contractor's project management	s	o	s	s	s			o	o	s	d	d
Member of development team	d	d	d	d	d	d	d	d	s	s	d	s
Testers and integrators	s	s	d	s	s	s	s	s	s	d	s	s
Third-party developers	d	s	s	s	d	o	s	s	o	o		o
Maintainer	d	s	d	s	s	s	s	s	s	s		
Analyst for performance	d	s	d	s	s	s	s	d	d	d		o
Analyst for safety	d		d	s	s	d	s	d	d	d		d
New stakeholder	x	x	x	x	x	x	x	x	x	x	x	s
Key: d = detailed information, s = some details, o = overview information, x = anything												

Table 1: NCORE Stakeholders and Architecture Information That They Find Useful

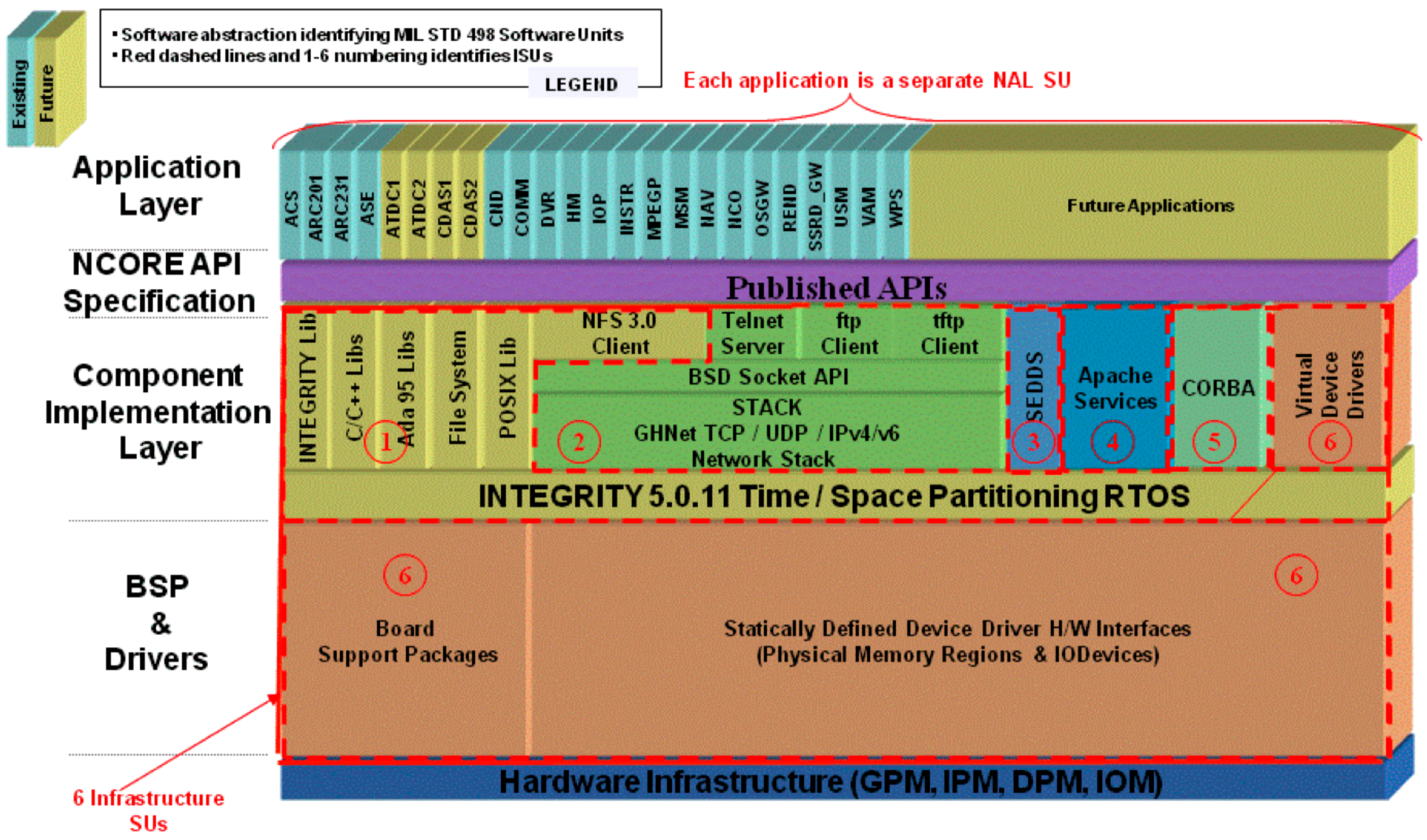


Figure 2: NCORE Software Unit Identification View

Frequent architecture information exchange among stakeholders fosters creativity and identifies continual opportunities for further exploitation of NCORE architecture capabilities at acceptable program risk levels.

Frequent Architecture Analysis and Improvement

Customer collaboration over contract negotiation is a well-known agile development practice. Agile design involves alternative analysis and trade-offs among evolving stakeholder concerns and among the significant design decisions made to address them. Periodic architecture evaluations enable and compliment continuous stakeholder education. Stakeholder knowledge sharing enables product improvement through exploitation of diverse viewpoints. Investing in frequent improvement, recognized as an “inspect and adapt” best practice, is about enhancing product capability based on direct stakeholder input and stated business goals.

A team of experts, composed of members of the Carnegie Mellon University SEI and the U.S. Army, evaluated the Apache Block III NCORE software architecture by applying the Architecture Tradeoff Analysis Method® (ATAM®) [4]. The evaluation team pulled information from the business management, architecture team, and stakeholders in a goal-directed fashion. The evaluation team analyzed the architecture with respect to the business drivers and stakeholders’ concerns about quality attributes [5]. Discovered risks served as feedback and guidance to the architects and business managers, and these risks became the focus of follow-on mitigation and improvement activities. This continual cycle influences and sustains evolution of the architecture and business drivers in a predictable manner. In addition to identifying risks, other important benefits to performing evaluations include clarification of stakeholder concerns about quality attributes and enhanced communication among the stakeholders [6].

Reports from the field validate and affirm the value of conducting periodic peer reviews during the design phase of the software development lifecycle [7, 8]. Apache quarterly software design reviews led by software architects and subject matter experts further enhance customer communications and stakeholder engagement. Customer comments from these reviews are iteratively incorporated in the design artifacts to continually improve product quality.

Strategies and Patterns for Sustained Evolution

Strategies for architectural governance and patterns for open systems and extensibility sustain long-term product evolution. Architecture evolution is enhanced through process-driven oversight centered on balancing flexibility and stability.

An Architecture Review Board (ARB) is tasked with governing the architecture to ensure that it evolves in a disciplined way. The ARB acts as an internal design review team of culturally diverse architects (differing viewpoints) and subject matter experts (specific domain knowledge) who are chartered to ensure minimally complex, consistent, and informed designs—all aimed at minimizing implementation time and cost and reducing the amount of rework.

The NCORE architecture is documented as a series of views [3] as required by evolving stakeholder needs. The software unit identification view is shown in Figure 2, and additional architecture documentation can be found in the work of Koontz [9, 10, 11].

Designing for extensibility promotes continued evolution and uses design patterns such as real-time technical metric reporting, publish-subscribe, client-server, and layering. Designing as an open system through nonproprietary application programming interfaces enables third-party software integration and the ability to move applications between CPUs during integration (to achieve load balancing, for example).

Prototyping and the Research & Development (R&D) Test Platform

The Apache Block III program continues to benefit from using and leveraging multiple prototyping activities. The MCAP II program, jointly funded by the AATD and Boeing, serves as an R&D flight test platform for evaluating emerging technologies targeted for production Apache. Targeted early prototyping significantly reduces program risk through technology culling and selective maturation. Examples of network-centric experimentation now transitioning into Apache Block III include tactical communication data links for H.264 video streaming, soldier radio waveform, wideband networking waveform, Link-16, and manned/unmanned teaming. Agility in the form of cross-functional teams and R&D-focused culture is paramount to the success of proof-of-concept technology demonstrations that further enable rapid system integration with acceptable program risk [12].

After emerging technologies are initially demonstrated and selected for product integration (new technology insertions), they are typically rapidly prototyped within the production environment. Prototyping at this later point in the lifecycle enables parallel requirements definition and software development, a recognized and proven agile practice. Agile application shortens overall integration lifecycles by merging requirements definition with software development, test, and integration process steps.

Applying Architecture Practices to Support Sustainable Delivery

Now that the five practices have been individually explained, they are applied in combination to demonstrate how they support the evolving system and delivery of new capabilities.

Evolving stakeholder needs and business goals identified through architecture evaluation lead to new requirements for selected capabilities. Initially, NCORE started with a primary focus on open-systems architecture, performance, and reliability and is now moving toward flight safety and extensibility (realizing the planned technology refresh must satisfy very long-term program objectives). These new requirements trigger the infrastructure and application-insertion timelines in response.

For example, during an ATAM evaluation, an exploratory scenario identified height-above-ellipsoid as a common platform technique for improving weapons-delivery accuracy that could be easily implemented. This kind of change is supported and sustained by the architecture practices working interactively and in unison in accordance with Table 2. Alternatively, first-time Joint Tactical Radio (JTR) Link-16 integration is viewed as a much more complex insertion; however, application of the practices is equally relevant.

The incremental/iterative development shown in Figure 1, now being deployed for JTR Link-16 integration, allows for focused and time-phased requirements/architecture analysis, code, test, and integration activities for complex and less defined function deployment based on stakeholder priorities and choices. Time-phased incremental/iterative development enhances testability due to incremental design verification and just-in-time architectural decision making that must be coordinated and scheduled.

Informed technology-insertion decision making applies to both JTR Link-16 and height-above-ellipsoid. It enables communication and understanding among multiple stakeholders regarding a specific concern and its business priority. From Table 1, the statement of work and performance specifications represent formal and binding contractual agreements that convey customer requirements to the architects and program management. These documents are pivotal for Apache because they represent coupling between prior architecture evaluation and contractual requirements.

Frequent architecture analysis and improvement is centered on brainstorming exploratory scenarios using agreed-upon architecture artifacts. The architectural views, shown in Table 1, provide evidence that quality attributes are being satisfied during consideration of height-above-ellipsoid specific concerns relative to business goals and priorities.

Strategies and patterns for sustained evolution are employed and enforced by the ARB to ensure architectural integrity across the lifecycle. Patterns for open systems and extensibility provide support for making and localizing architecture change. Insertion agility is the result of identifying required architecture changes and employing an agile just-in-time methodology (e.g., adding secure socket library in support of Link-16 integration).

The NCORE architecture was first-flight proven in 2004, using the MCAP II Prototyping and R&D Test Platform. Initial NCORE demonstration validated the architecture capability to rapidly integrate new technologies and is the keystone open systems architecture enabler for the Apache Block III program. Additionally, network-centric experimentation has led to the customer's decision to choose Apache Block III as the first JTR Link-16 integration platform.

Based upon discussions with the stakeholders, height-above-ellipsoid is being provided in a future enhancement statement of work and will soon be implemented and deployed. JTR Link-16 software development and integration continues to mature and evolve with the delivery of incremental capabilities.

Agile Development and Software Architecture Enablers

Table 2 characterizes the five architecture practices using established criteria from agile software development and software architecture fundamentals, including response to change, customer collaboration, quality attributes, and architecture, so they can be applied to benefit the development of other software-

	Response to Change	Customer Collaboration	Quality Attributes	Architecture
Incremental / Iterative Development	Necessary processes are identified to respond to change	Functional requirements are communicated with focused criteria and business priority	Quality attribute requirements are defined and tied to business goals	Timeline of critical architectural decisions is clear and scheduled
Informed Technology-Insertion Decision Making	Dynamic environment and changing requirements are understood	Effective customer communication channels manage expectations	The importance of quality attribute requirements is understood	Architectural issues (e.g., technical debt) are tracked and managed
Frequent Architecture Analysis and Improvement	Waste is identified and trade-offs are managed (e.g., technical debt and defects)	Artifacts to keep multiple stakeholders informed are agreed upon and produced effectively	Quality attribute requirement analysis is in place and used to predict system properties	Evidence is provided that the architecture satisfies quality attribute requirements
Strategies and Patterns for Sustained Evolution	Impact of uncertainty on the project is acknowledged	Technology insertions are driven and targeted by the user	Quality attribute design is aligned to lifecycle maintenance	Just-in-time architecting enables technology-insertion agility
Prototyping and R&D Test Platform	High-potential technologies are identified to respond to change	Pipeline of emerging technologies and technology insertions are mapped to evolving business goals	Measurement environment is in place to monitor the implemented system quality and done criteria	Obsolescence risk management occurs via prototyping of newest avionics technologies (multicore processors)

Table 2: Mapping of Practices to Agile and Architecture Criteria

reliant systems across different domains. These criteria provide a quick look into the application of the practices and associated risks to enabling the ability to sustain software development and delivery at the expected velocity (pace) for large-scale, complex, multiyear projects [13].

Key Takeaways

• For software-intensive, multiyear projects, agile development, which is focused on rapid, short-term deliverables, must be complemented by sustainable architecture practices that ensure the incremental delivery of capabilities over the extended lifecycle of the product.

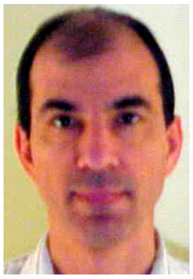
• Software architecture best practices support sustainable software delivery by leveraging established criteria from agile software development and by applying software architecture fundamentals that include response to change, customer collaboration, quality attribute trade-offs and analysis, and architecture governance. These practices are interrelated and interact to provide sustainable delivery of quality products.

• Architecting with agility can be applied across the lifecycle to continuously develop, deliver, and enhance software-reliant systems. ◈

ABOUT THE AUTHORS

Ronald J. Koontz is a Boeing Company Technical Fellow with more than 20 years of expertise in real-time embedded systems. He is an SEI-certified ATAM evaluator and holds a Boeing Software Architect Certificate. He has served as a software architect on U.S. Army Apache Attack Helicopter Mission Processor projects since 2003. Prior to Apache programs, he led the design, implementation, and field testing of multiple software-intensive projects for The Boeing Company, Phantom Works.

Phone: 480-891-2065
E-mail: ron.j.koontz@boeing.com



Robert L. Nord is a senior member of the technical staff at the SEI and works to develop and communicate effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books Applied Software Architecture and Documenting Software Architectures: Views and Beyond, published by Addison-Wesley, and lectures on architecture-centric approaches.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA
Phone: 412-268-1705
Fax: 412-268-5758
E-mail: rn@sei.cmu.edu

REFERENCES

1. Highsmith, J. A. Agile Project Management: Creating Innovative Products. 2nd ed. Boston: Addison-Wesley Professional, 2009.
2. Brown, N., R. Nord, and I. Ozkaya. "Enabling Agility Through Architecture." CrossTalk 23.6 (November/December 2010): 12-17.
3. Larman, C. and V. R. Basili. "Iterative and Incremental Development: A Brief History." IEEE Computer, 36(6), 2003: 47-56.
4. Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures: Views and Beyond. Boston: Addison-Wesley, 2003.
5. Clements, P., R. Kazman, and M. Klein. Evaluating Software Architectures: Methods and Case Studies. Boston: Addison-Wesley, 2002.
6. Ozkaya, I., L. Bass, R. Sangwan, and R. Nord. "Making Practical Use of Quality Attribute Information." Spec. issue of IEEE Software 25.2 (March/April 2008): 25-33.
7. Nord, R. L., J. Bergey, S. Blanchette Jr., and M. Klein. Impact of Army Architecture Evaluations (CMU/SEI-2009-SR-007). Software Engineering Institute, Carnegie Mellon University, 2009. <http://www.sei.cmu.edu/library/abstracts/reports/09sr007.cfm>
8. Edmondson, J. S., E. Lee, and C. G. Kille. "A Light-Weight Architecture Trade Off Process Based on ATAM." SEI Architecture Technology User Network (SATURN) Conference. Sheraton Station Square, Pittsburgh. 14-16 May 2007. <http://www.sei.cmu.edu/library/abstracts/presentations/ATO-Lite-for-SATURN-2007-2.cfm>
9. Forstrom, H. "Inexpensive ATAM-Peer Review Detects and Fixes Architecture Problems Early." SEI Architecture Technology User Network (SATURN) Conference. Radisson Green Tree, Pittsburgh. 30 April-1 May 2008. <http://www.sei.cmu.edu/library/abstracts/presentations/ATAM-peer-review-SATURN-2008.cfm>
10. Koontz, R. "Mission Processor Software Open Systems Architecture for the Apache Helicopter." Proceedings of the 63rd American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2007. 2253-2261.
11. Koontz, R. "Apache Mission Processor Software Architecture: Architectural Approaches." Proceedings of the 64th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2008. 1507-1514.
12. Gannon, S.P. and R.E. Speir. "US Army Aviation Network Technologies Demonstrated at JEFX'08." Proceedings of the 65th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2009. 812-823.
13. Koontz, R. "Apache Mission Processor Software Architecture: Architectural Decisions." Proceedings of the 65th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2009. 766-774.

WANTED

Electrical Engineers and Computer Scientists
Be on the Cutting Edge of Software Development

The Software Maintenance Group at Hill Air Force Base is recruiting **civilian positions** *(U.S. Citizenship Required)*. Benefits include paid vacation, health care plans, matching retirement fund, tuition assistance and time off for fitness activities. **Become part of the best and brightest!**

Hill Air Force Base is located close to the Wasatch and Uinta mountains with many recreational opportunities available.

Send resumes to:
phil.coumans@hill.af.mil
or call (801) 586-5325
Visit us at:
http://www.309SMXG.hill.af.mil

Architectural Tactics to Support Rapid and Agile Stability

Felix Bachmann, SEI
Robert L. Nord, SEI
Ipek Ozkaya, SEI

Abstract. The essence of stability in software development is the ability to produce quality software with infrastructure that will meet long-term business goals. The essence of rapid and agile development is the ability to deliver capabilities quickly based on customer priorities. Stability often requires cross-functional analysis and infrastructure support that will build foundational technology for the capabilities to stand on, which takes time and resources. But today's organizations must attend to both agility and enduring design. This article presents three tactics that support rapid and agile stability: aligning feature-based development and system decomposition, creating an architectural runway, and using matrix teams.

Today's organizations must design, develop, deploy, and sustain systems for several decades and manage system and software engineering challenges simultaneously; neither agility nor attention to enduring design can be dispensed with [1]. Systems developed at such a scale go through several funding and review cycles and are typically meant to operate for several decades, so longevity and stability are key goals. Shrinking defense budgets and continued demand for new capabilities add pressure to use rapid and agile development and deployment. All software-intensive systems relevant to the DoD fit this description due to the existing acquisition processes and the need to support the systems for extended periods of time in the field in order to manage costs.

This article presents the lessons we learned from our interactions with teams and individuals in the roles of Scrum master, developer, project manager, and architect on projects from organizations that develop embedded real-time software or cyber-physical systems. We observed that the following symptoms surface from the lack of stability to sustain rapid and agile software development:

• Scrum teams, product development teams, component teams, or feature teams spend almost all of their time fixing defects, and new capability development is continuously slipping.
• Integration of products built by different teams reveals that incompatibilities cause many failure conditions and lead to significant out-of-cycle rework.
• Progress toward meeting milestones is unsatisfactory because unexpected rework causes cost overruns and project completion delays.

If part of the problem is due to a mismatch, in which architectural decisions fail to support business goals including agility, then part of the solution is to introduce architecture in a coherent way. This does not mean to fall back to building all of the architecture in advance of development. It is possible to introduce the concept of incremental architectural development into the agile development approach.

The solution can be described as a desired software development state that enables agile teams to quickly deliver releases that stakeholders value. When a product development starts, this desired state does not necessarily exist. The teams themselves typically define the desired state. It is their vision of the ideal development infrastructure that they would like to work with. This is a state in which platforms and frameworks, as well as tool environments and processes, exist that support efficient, independent development of user features.

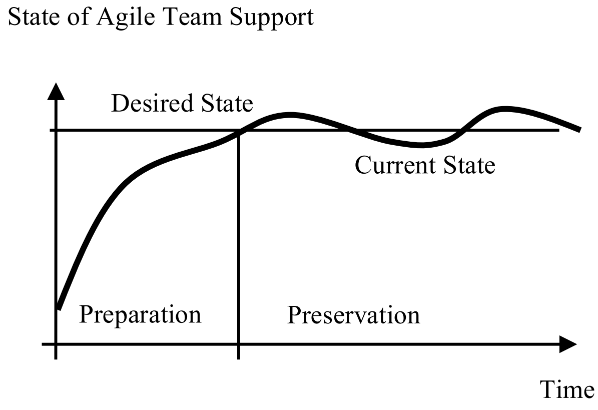


Figure 1: Infrastructure support for agile development teams over time

If the desired state does not yet exist, the agile teams first go through a preparation phase (Figure 1). The goal of this phase is to do all the preparation required to move to the desired state. During this phase, the teams can deliver releases, but they will not deliver as much value to the stakeholders as they would if they were in the desired state. This is because many tasks focus on maturing the platforms, frameworks, and tool environments.

Once they have achieved the desired state, agile teams enter the preservation phase. The goal of this phase is to maintain the desired state by dealing with technical debt, changing requirements, and new technologies. In this phase, it is critical to neither over-optimize the development infrastructure nor to quit working on it. Over-optimization incurs cost without much benefit for the

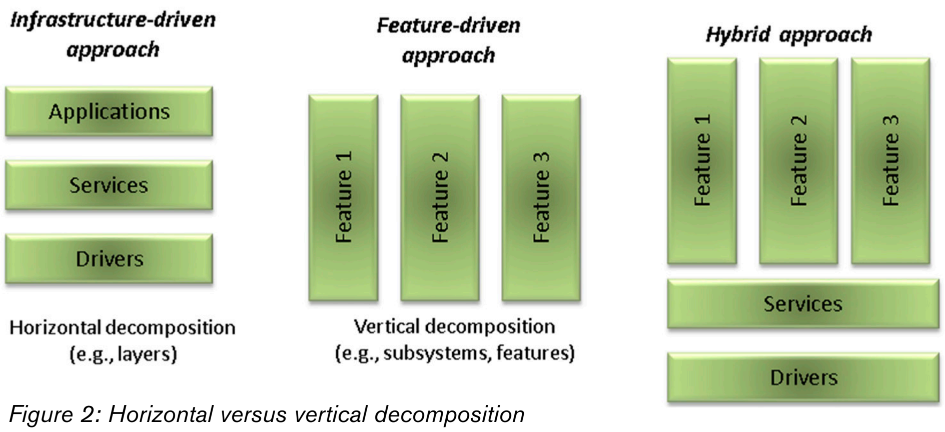


Figure 2: Horizontal versus vertical decomposition

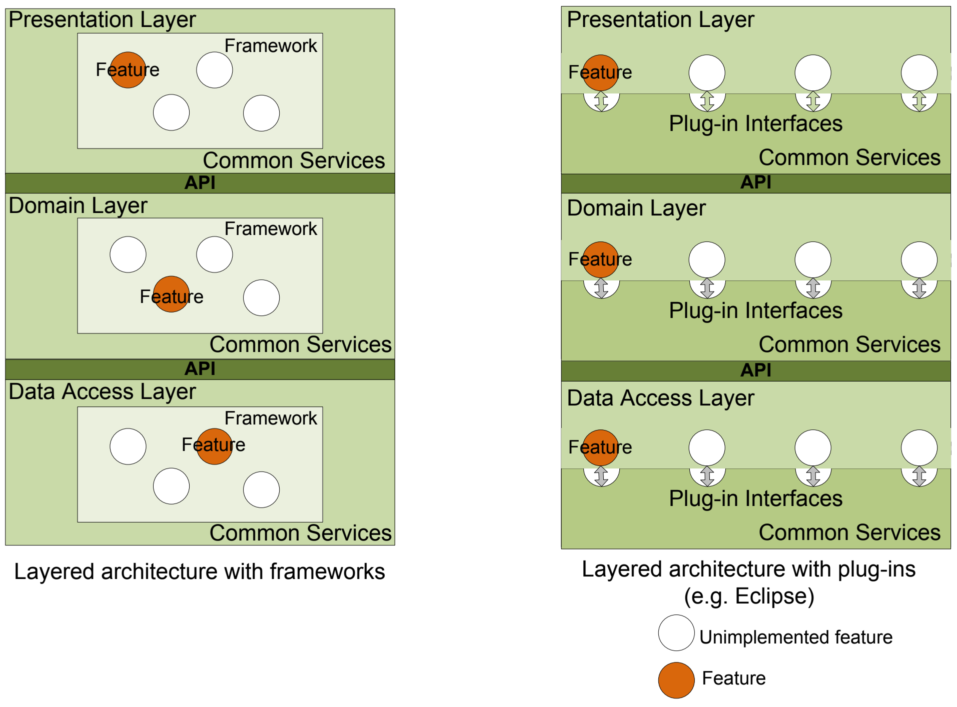


Figure 3: Layered architecture supporting feature-based development

teams (waste), while not evolving the infrastructure slows down the teams over time (also waste).

To achieve the desired productivity, different team structures must align with the desired state. Three distinct tactics can help teams move to and maintain the desired state: (1) Align feature-based development and system decomposition; (2) create an architectural runway; (3) use matrix teams. We describe these tactics in the following sections and then show how they can be used together to deliver software with agility when the agile team pays explicit attention to the underlying infrastructure to support that agility.

Aligning Feature-Based Development and System Decomposition

A common approach in agile teams is to implement a feature (or user story) in all of a system's components. This gives the team the ability to focus on something that has stakeholder value. The team controls every piece of implementation for that feature; therefore, they do not have to wait until someone outside the team has finished some required work. We call this vertical alignment because every component of the system required for realizing the feature is implemented only to the degree required by the team. System decomposition could also be horizontal based on the architectural needs of the system, focusing on common services and variability mechanisms (reuse). Figure 2 illustrates these two different approaches and how they can coexist. The goal of creating a feature-based development and system decomposition approach is to provide flexibility in aligning teams horizontally, vertically, or in combination and to ensure progress by minimizing coupling.

Although organizations create products in very different domains (from embedded systems to enterprise systems), similar architectures emerge when development teams need to support rapid and agile stability. The teams create a platform containing commonly used services and development environments either as frameworks or platform plug-ins to enable fast, feature-based development.

Figure 3 shows two generic examples of such architectures. Here we have an architecture consisting of three layers. Every layer has either a framework (left side) or a plug-in interface (right side) defined that implements the control logic of that layer. Every layer also has a collection of services that provide common functionality. To develop a feature, the agile team implements only the logic of that feature in each layer using the frameworks or plug-in interfaces. This focuses the development on what is needed for the feature implementation. The frameworks and common services already include all the logic of how to integrate new pieces of code into the system, such as by using intra-layer communication protocols. This type of architecture also minimizes the dependencies between different feature implementations so that different teams can implement features without coordination, further enhancing stability, and rapid development as users need new features.

Not every project has an existing architecture that would support feature-based development from the start. Since it is a fairly difficult task to define the appropriate APIs, common services, and plug-ins/frameworks up front, teams usually choose an evolutionary approach.

In a large multiyear client-server development project, we observed extensive use of aligning feature-based development and system decomposition to manage agile stability. Using Scrum, 25 teams participated in the development effort. Some of the teams were colocated;

other teams were located in different countries. There were teams responsible for applications, others for the platform, and others for architecture and quality assurance. In this project, the teams had a platform-oriented focus at the beginning during the preparation phase and switched to a more application-oriented focus later in the preservation phase, reflecting the change of focus in their architecture with a hybrid approach of vertical and horizontal decomposition.

Creating an Architectural Runway

An architectural runway can help provide the degree of architectural stability required to support the next iterations of development [2]. This stability is particularly important to the successful operation of multiple parallel Scrum teams. Making architectural dependencies visible allows them to be managed and for teams to be aligned with them. The runway supports the team decoupling necessary to allow independent decision-making and reduce communication and coordination overhead.

During the preparation phase, agile teams build a runway of infrastructure sufficient to support the development of features in the near future. Product development using an architectural runway most likely occurs in the preservation phase. Dean Leffingwell explains that intentional architecture is one of the key factors to successfully scale agile [2]. Building and maintaining the architectural runway puts in place a system infrastructure sufficient to allow incorporation of near-term high-priority features from the product backlog. This preparation allows the architecture to support the features without potentially creating unanticipated rework by destabilizing refactoring.

Larger systems (and teams) need longer runways. Building and rearchitecting infrastructure takes longer than a single iteration or release cycle. Delivery of planned functionality is more predictable when the infrastructure for the new features is already in place. This requires looking ahead in the planning process and investing in architecture by including infrastructure work in the present iteration that will support future features that the customer needs.

The architectural runway is not complete. The runway intentionally is not complete because of an uncertain future with changing technology and requirements. This requires continuously extending the architectural runway to support the development teams.

We observed one Scrum team that had already benefitted from an existing and proven platform. The architect of that platform was the driver and Scrum master of the development team. The team added features (vertical alignment) to the product quickly on top of the existing infrastructure while the architect, with temporarily assigned team members, implemented additional platform changes required for future features (horizontal alignment of the system into layers). The growing platform provided them with a runway sufficient to build the desired functionality for the complex embedded, real-time system environment.

Using Matrix Teams and Architecture

In its simplest instantiation, a Scrum development environment consists of a single colocated, cross-functional team with the skills, authority, and knowledge required to specify requirements and architect, design, code, and test the system. As systems grow in size and complexity, the single-team model may no longer meet development demands.

A number of different strategies can be used to scale up the overall development organization while maintaining an agile Scrum-based development approach. One approach is replication, essentially creating multiple Scrum teams with the same structure and responsibilities, sufficient to accomplish the required scope of work. This is the approach advocated by the Scrum Alliance. Some organizations scale Scrum through a hybrid approach. The hybrid approach involves Scrum team replication but also supplements the cross-functional teams with traditional functionally oriented teams. An example would be using an integration-and-test team to merge and validate code across multiple Scrum teams. (Note that Scrum purists would most likely label the hybrid approach an example of “Scrum But.”)

In general, we recognized two criteria used to organize the teams. The first criterion is organizing the teams either horizontally or vertically, assigning different teams the responsibility for either components (horizontal) or features (vertical). The second criterion is assigning the teams responsibilities according to development phases.

Aligning the teams horizontally is a good idea during the early stages of the preparation phase, while vertical alignment works well during the preservation phase. Between those two states, we find matrix structures in which the teams are aligned either horizontally or vertically while some members within those teams have the opposite responsibilities [3].

In another multiyear project, we observed two distributed Scrum teams that worked in an environment where project management and quality assurance were more waterfall oriented, causing tension because the teams delivered incremental results that were not aligned with the overall waterfall approach. When the organization decided to switch the project management and quality assurance groups to an agile approach, the architects integrated into the Scrum teams. This helped them achieve a matrix team structure to manage responsibilities for developing components and features effectively.

Applying the Tactics in Concert

Let us see how these tactics work together to provide infrastructure support for agile development over time. In Figure 4, we marked points in the state of the product development that we presented in Figure 1. These points are not exact. We use them here to give an estimate of when certain development strategies and team structures make sense.

Here we assume that when the product development starts no (or minimal) support for agile teams is available. This means there is no existing platform or frameworks, and the tool environment may not be established yet.

At the starting point (point A in Figure 4), it makes sense to organize the teams horizontally. Most of the teams’ responsibilities involve getting the supporting infrastructure to a point at which feature development can start. During this period, team members will create a rough sketch of the architecture, make technology

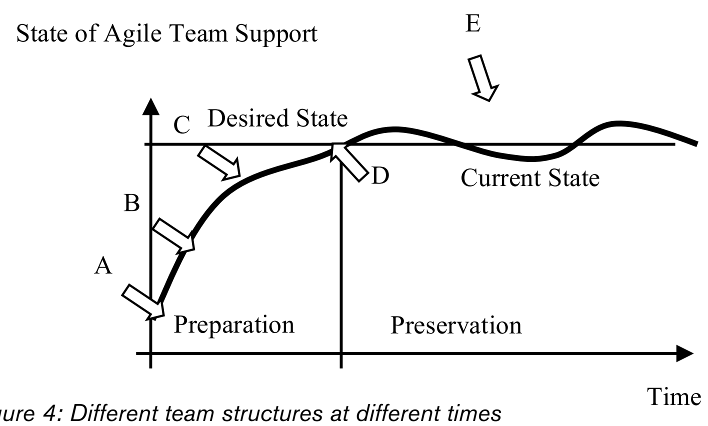


Figure 4: Different team structures at different times

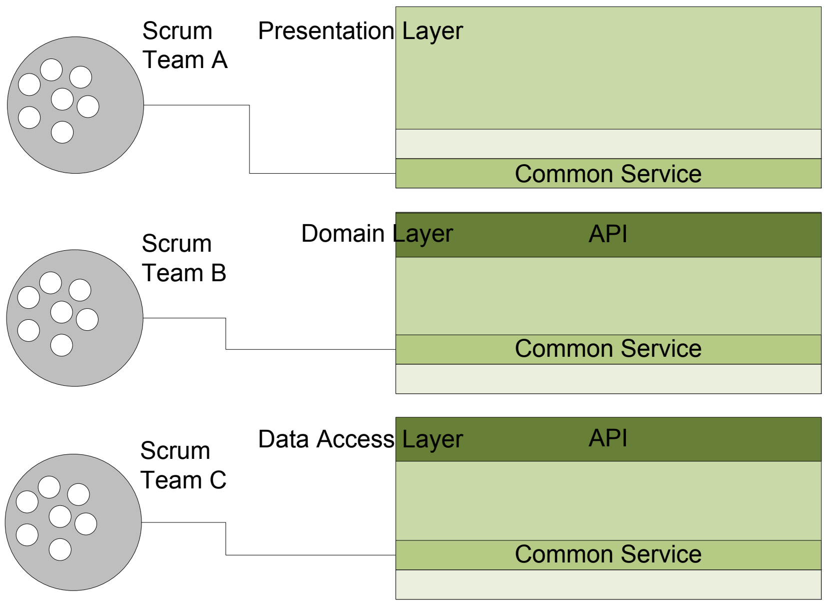


Figure 5: Layered architecture implemented with some common services and APIs

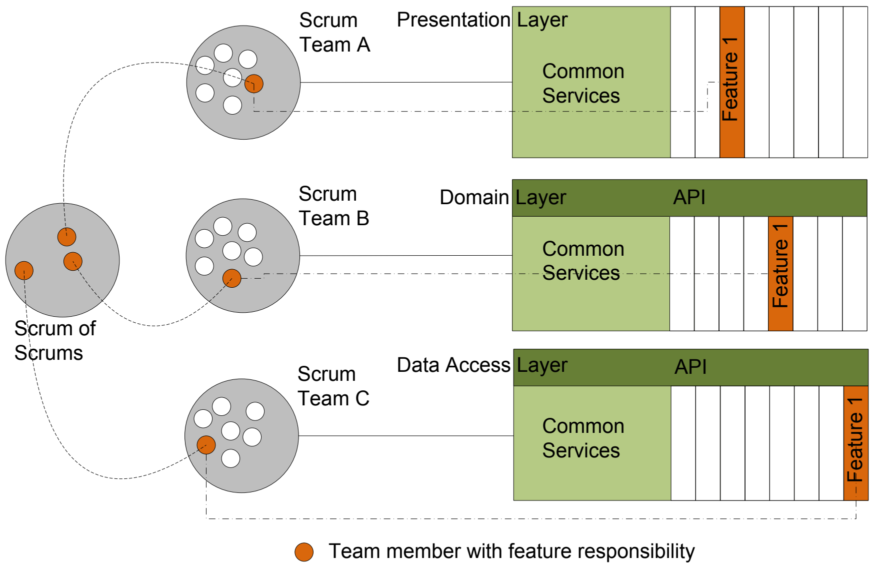


Figure 6: Feature development in parallel on top of a skeleton system; different teams assigned to layers (horizontal alignment) with some team members assigned to implement features

decisions, establish the tool environment, and so on. Typically, teams will use a small subset of basic user features (user stories) to guide the creation of the development infrastructure consisting of basic common services and APIs of the layers. Those basic features may not be implemented during this phase. Sometimes the resulting product is called a skeleton system. The result of this phase of the project is a first version of a platform that is good enough to be used to develop the first features (see Figure 5 for a notional example), as described previously in the feature-based development section.

As soon as the most important interfaces are defined, some team members can start developing features (point B in Figure 4). We now start seeing a matrix organization. During this time, most team members will still have component-oriented responsibilities. Therefore, the teams are still horizontally organized, but some team members now have the responsibility to start implementing features using the development infrastructure built so far. This pressures teams to start organizing themselves according to features and implementing them on top of the skeleton system. In a Scrum of Scrums, the team members assigned to implement features coordinate with each other to ensure on-time delivery of the features. This helps stabilize the interfaces and provides the first ideas for implementation frameworks that would support feature development (see Figure 6).

With the interfaces getting more stable, the time comes to switch most of the teams to vertical (feature-oriented) development (point C in Figure 4). In this situation, we found that some team members still had horizontal responsibilities because the development infrastructure was far from complete and teams implemented common services as well as framework and interface enhancements continuously (Figure 7).

In doing so, the teams get closer to the desired state in which they can focus on feature development (point D in Figure 4). Now the architecture has reached a level of maturity and teams have the necessary infrastructure to implement features quickly. The feature-based development aims to better manage and demonstrate end-to-end features and provides the ability to assign features to teams without too many dependencies between them. Especially in a context where there is a high number of Scrum of Scrums and many customer-facing features, this approach can help align the teams with the system structure. In one example where we observed this approach, the number of

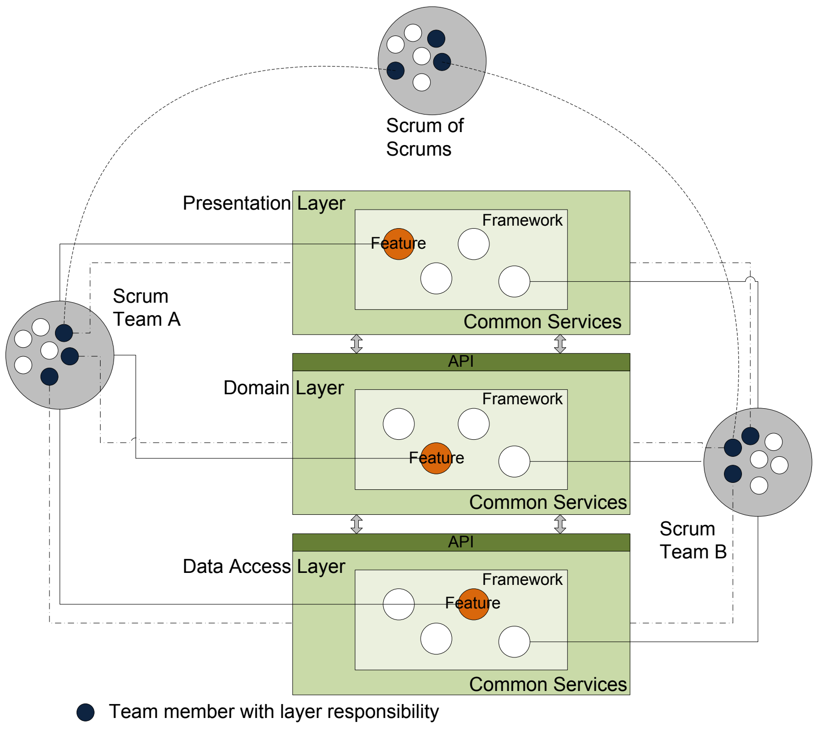


Figure 7: Different teams assigned to features (vertical alignment) with some team members assigned to keep layers and frameworks consistent

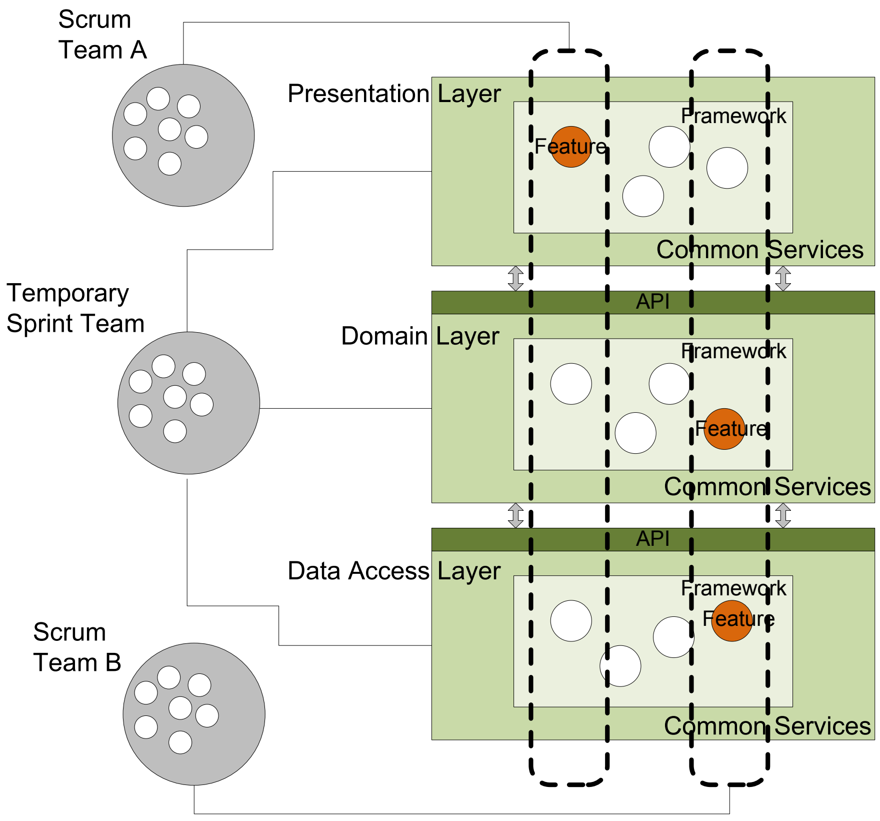


Figure 8: Different teams assigned to features (vertical alignment), with a temporary team assigned to prepare layers and frameworks for future feature development

teams participating on a Scrum of Scrums was 25; hence, the feature-based development and system decomposition approach helped separate team dependencies at the feature level.

At this point, the preservation phase starts. Few team members, if any, will have horizontal responsibilities. The goal of the preservation phase is to continue to build the next piece of the architectural runway that the system will need in the future. Every product development has to cope with changing requirements and new technologies (point E in Figure 4).

In one project we analyzed during the preservation phase, the product architect had the responsibility to look ahead and decide what the system would need in the future. He then assembled a team, and in a sprint they developed the next piece of the runway. After the sprint, the team was dissolved. Meanwhile, all the other teams were still organized vertically, developing features for their customer (Figure 8).

Takeaways

Achieving rapid and agile stability for fast yet steady development is a matter of aligning the right practices with the needs of the development effort. No one tactic can bring success to any project. The principles of both agile software development and software architecture provide improved visibility of project status and better tactics for risk management in order to create higher quality features within the required time frames and optimum use of resources. In this article, we described three tactics: aligning feature-based development and system decomposition, creating an architectural runway, and using matrix teams and architecture. Harmonious use of these tactics is critical, especially in DoD-relevant systems that must be in service for several decades, that are created by several teams and contractors, and that have changing scope due to evolving technology and emerging new needs.

Disclaimer:

This material is based upon work funded and supported by the DoD under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research, and development center.

ABOUT THE AUTHORS

Felix H. Bachmann is a senior member of the technical staff at the Software Engineering Institute (SEI) in the Architecture Centric Engineering Initiative. He is coauthor of the Attribute-Driven Design Method, a contributor to and instructor for the Architecture Tradeoff Analysis Method® Evaluator Training, and coauthor of Documenting Software Architectures: Views and Beyond. Before joining the SEI, he was a software engineer at Robert Bosch GmbH in Corporate Research for small and large embedded systems.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA
E-mail: fb@sei.cmu.edu
Phone: 412-268-6194

ABOUT THE AUTHORS (continued)

Robert L. Nord is a senior member of the technical staff at SEI and works to develop and communicate effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books Applied Software Architecture and Documenting Software Architectures: Views and Beyond and lectures on architecture-centric approaches.

E-mail: rn@sei.cmu.edu
Phone: 412-268-1705

Ipek Ozkaya is a senior member of the technical staff at SEI and works to develop empirical methods for improving software development efficiency and system evolution with a focus on software architecture practices, software economics, and requirements management. Her latest publications include multiple articles on these subjects focusing on agile architecting, dependency management, and architectural technical debt. Ozkaya serves on the advisory board of IEEE Software.

E-mail: ozkaya@sei.cmu.edu
Phone: 412-268-3551

REFERENCES

1. Brown, N., R. Nord, and I. Ozkaya. "Enabling Agility Through Architecture." CrossTalk 23.6 (2010): 12-17.
2. Leffingwell, D. Scaling Software Agility. Upper Saddle River, NJ: Addison-Wesley, 2007.
3. Reinertsen, D. G. Managing the Design Factory: A Product Developer's Toolkit. New York: The Free Press, 1997.

NAVAIR Vision Statement:
"Sailors and Marines, Armed and Operating with Confidence"

Because we develop, deliver, and sustain aircraft, weapons, and systems—on time, on cost, with proven capability and reliability—so they cost effectively succeed in every mission and return home safely.

NAVAIR Process Resource Team (PRT)
(760) 939-6226

Optimized Project Management of Systems and Software Projects

Denton Tarbet, Galorath Incorporated

Abstract. Successful software projects demonstrate good project management methods incorporating modern processes and effective utilization of metrics. This paper extends the application of control theory methods to project management decisions to include optimization of systems trade-offs to improve project performance. With the application of feedback control theory to the management decision process, future project control decisions result in changes that improve performance of the project process.

Issue

Systems and software projects traditionally experienced some degree of project failure. In this context, failure is defined as a project that demonstrated a failure to match (within a reasonable tolerance) the expected outcome. Studies such as the project data collected by the Standish Group in its annual Chaos reports [1] on software projects indicate improvement in software project success but the failure to meet project objectives is still high. Depending on the data referenced, there is a 50% to 80% probability that a systems and software project will:

1. **Require significantly more time than planned.**
2. **Cost significantly more than budgeted.**
3. **Deliver significantly less functionality than originally expected.**

In addition, project problems seem to demonstrate a direct correlation between the size of the project and the degree of project failure. That is, the larger the project, the longer the planned duration, and the greater the likelihood that the project will encounter at least one of the conditions. If we define success as achieving or exceeding expectations; i.e., success occurs when the actual outcome matches (within a reasonable tolerance) the expected outcome [2]. It follows that project success implies the need for a roadmap or plan that ensures meeting objectives within a reasonable confidence level. Software project performance can be improved with the application of effective management control during execution [3]. This paper proposes the application of classical control theory overlaid with optimum process control to establish methods to ensure the project maximizes potential of success.

System and Software Project Management and Control

Project management has been defined as, "A discipline that employs skills and knowledge to achieve project goals through various project activities." It involves planning, organizing, leading, and controlling costs, time, risks, project scope, and quality [4]. Systems and software project management in the context of this paper fulfills that definition. It is critical for success that we not only have a plan to implement, but also a way to lead and control to ensure a "best" project result.

Software projects can be considered as a system process with project management which:

1. **Follows a detailed plan.**
2. **Can be evaluated in process with process metrics.**
3. **Includes feedback to project management.**
4. **Incorporates feedback response into control decision process.**
5. **Applies process control "during the process" to reduce the risk of not meeting project objectives. That is to "optimize the project performance" and maximize probability of project success.**

The methods suggested to provide project management are aligned with classical control systems theory. Control systems theory provides a rigorous framework for analyzing complex feedback systems. We are all familiar with real-time control theory such as when driving a car. The driver constantly monitors the car's position in the lane. The process of monitoring the actual position against desired position and making steering adjustments is similar to tracking and controlling a software project. Similar mathematics apply, so that optimal control processes and modeling of classical control theory can be applied to systems and software project management [5].

Control Theory Applied to Systems and Software Project Management

Control systems in the broadest sense are critical in a wide range of applications. In manufacturing processes as simple as machining parts, the machines are set up, the manufacturing process is started, and output parts are regularly measured to ensure meeting specifications. Any changes of the manufactured parts away from specifications will be noted by the metrics being collected. As changes are noted, corrective action is taken to ensure the manufactured parts remain within specifications. Control theory is normally based on the principle of feedback:

1. **A desired objective is defined for the system.**
2. **Resources are applied, raw material is processed.**
3. **Sensors are used to collect metrics.**
4. **Metrics are analyzed to determine if the system is meeting its objectives.**
5. **Math models of the "real world" system are typically used to choose corrective actions from a set of possible actions.**
6. **Corrections are made to the input in order to improve the performance of the system with respect to the "desired objective."**

When control theory is applied to a dynamic system, it is critical to include a method to evaluate alternatives in the process of defining the "best" corrective actions. System dynamics provides the framework for modeling the systems and software process [5]. Within the overall system model, a critical decision tool is the method utilized to evaluate alternative actions. The application of control theory requires:

1. **A defined system objective, for optimization the objective is referred to as a "cost function."**
2. **A control law—i.e. the initial software development plan.**
3. **Measurement of effectiveness accomplished provided by the set of software metrics collected.**
4. **A method to evaluate alternative corrective actions.**

For the control process to be an "optimal control" process there must be a well defined "cost function." The cost function provides the basis for decisions reflected into the application of the "control corrections." The cost function or project goal presents a critical initial decision for the project. The goal can be to optimize, or minimize the difference between project cost and budgeted cost while monitoring schedule. The goal could be to minimize the difference between the actual project schedule and the planned schedule while monitoring cost. Applying classical control theory to the project management of systems and software projects can:

1. **Optimize performance for cost within an acceptable schedule.**
2. **Optimize performance for schedule within an acceptable cost.**
3. **Optimize performance for functional capability with acceptable cost and schedule.**

It will be a program management issue to define the objectives and resolve any conflicting goals for the project.

Systems and Software Projects

Considering the systems and software project, for a typical project with a defined beginning and end to develop a product a detailed project plan is developed. Project management assigns resources and initiates the project. Past projects have often demonstrated a process reflective of Figure 1.

The process runs according to the plan but when disturbance forces impact the process, there is no well-defined method to react to the impacts and correct the controller functions to improve performance. The goal of applying control theory is to implement the process as in Figure 2.

The process of Figure 2 includes the action to determine corrective actions to insert into the controller. For software process, a parametric effort and risk estimation tool, such as SEER-SEM [6], provides an effective method to identify and evaluate alternative actions that can be applied to improve project performance against the objective.

Project management implements methods to regularly review the process metrics and identify the requirement to implement corrective actions. Experienced project managers identify alternative actions to improve project performance. However, without

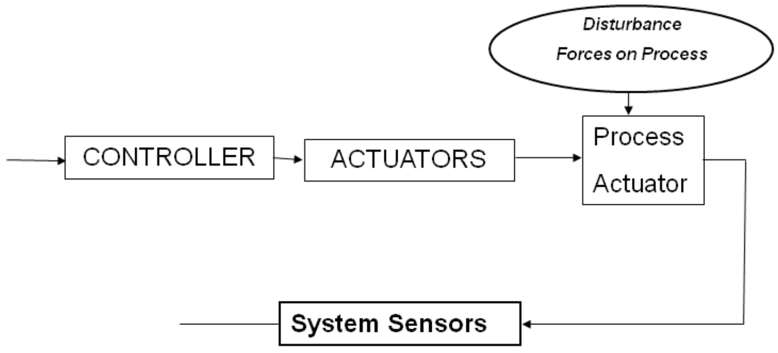


Figure 1

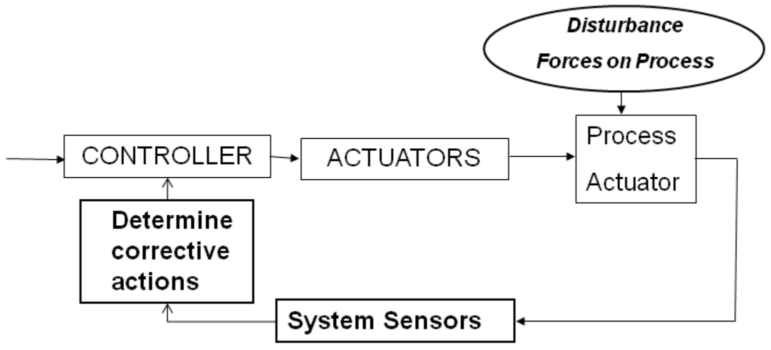


Figure 2

an effective tool such as a parametric software process model, it is difficult to identify reasonable project corrective actions that yield the best ROI. Occasionally, program management will suggest adding people to a project that is behind in schedule, or proposing that the team "work harder" which normally means extended overtime. However, it is easy to demonstrate the realities of Brook's law (i.e. adding people to a late project normally makes it later) with a parametric model that is set to minimum time estimation. The goal for project management should be to identify potential corrective actions that can be implemented by the project management and not as a function of asking the team to "work harder." Actions that can be implemented by project management that will affect productivity on the project include:

1. **Reducing the requirements changes that impact the program.**
2. **Reducing the level of documentation required for the delivered product.**
3. **Updating software development systems to reduce the rehost from development system to target system.**
4. **Providing effective total software development tool sets (assuming there is sufficient time to incorporate the new tools without impacting the project).**

A critical requirement for project management is the collection of a regular basis of effective project metrics. The measurement on the process is not of value if it is implemented as a "check-thebox" process rolled out to satisfy a scheduled review or process improvement assessment. The metrics should be collected to support the "feedback control" of the project. Measurement must provide real information to support critical project and organizational business and technical decisions.

Applying classical control theory implies a feedback control loop that necessitates continual collection of metrics reflecting performance of the project development process. In the situation of optimally controlling a software development process the feedback control system can effectively incorporate the parametric model that should provide a best possible representation of the real world. The model will provide a basis to evaluate alternative control applications.

Applying the control theory construct to a software project would result in a process that can be represented by Figure 3.

In this process the system requirements that provide the "desire" are utilized initially to develop an estimate of effort, schedule, and risk. The project plan (schedule with allocated resources) is developed and implemented. Project direction is the responsibility of the project manager and the software development process is initiated. At regular intervals, process metrics are collected and provided as performance feedback into the estimation process. An effective parametric model which is being maintained, i.e. updated with the metrics data, can be used to evaluate the cost to complete (both in effort and schedule) and provide that input to the project manager for decisions with respect to implementing corrective action. With a decision to implement corrective action, the estimating activity takes on an expanded role of reviewing suggested process actions to provide an estimate of the ROI for each alternative action considered. With the selection of corrective action(s), a parametric model software project estimation model is updated to match as closely as possible project status and utilized to provide a revised estimate of effort, schedule, and risk between alternative proposed actions in order to establish a consistent ROI between alternative actions. The use of a parametric estimation tool provides a consistent method to evaluate between alternate actions that are being considered.

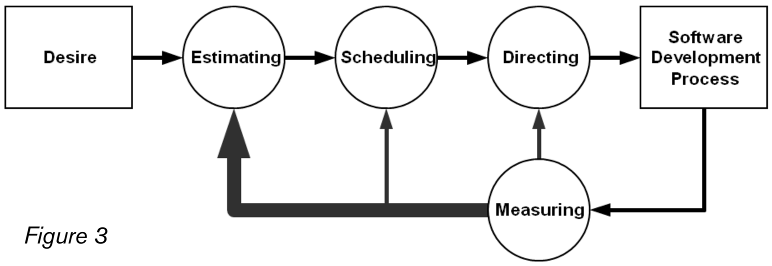


Figure 3

Without a parametric model, many software organizations incorporate one or more of the following to evaluate software project alternatives: 1) ballpark/rough order of magnitude engineering estimate, 2) top-down/constraint-driven estimating, and/or 3) bottom up/design-driven estimating. Despite the difference in planning methodologies, what all of these approaches have in common is that they tend to be executed as labor intensive practices dependent on the availability of scarce, over-committed personnel and manual or minimally-automated processes such as spreadsheets and other homegrown tools. Such practices are, by their nature, inconsistent, unpredictable, and highly susceptible to human error. Individual planners possess diverse capabilities and project histories; they may be overly optimistic or pessimistic; they may be influenced by internal politics or other factors unrelated to the project; or they may simply overlook some of the less obvious project elements.

Applying an established, proven estimation process that is well integrated with development processes helps ensure that project plans are credible and achievable, meet, or exceed customer expectations [7].

Parametric estimation models have been shown to provide a tool to develop the initial estimate and project plan as well as an effective means to evaluate project alternatives to optimize performance against goals [8]. With the establishment of a "high fidelity" parametric model during project conception, it only becomes necessary to maintain the model during project execution by updating the model effectively from the regularly scheduled metrics collected on the project. With parametric project modeling, it is expected that the standard metrics collected to support an Earned Value Management (EVM) process will provide sufficient update and calibration metrics for the model.

Performance-based Project Management

Central to Performance-based Project Management (PBM) is the goal of reducing risk in the forecasted effort and schedule of the final project outcome at each stage of the project. Application of PBM requires the establishment of a project baseline and estimate. That baseline will be considered the starting point for PBM. At project stages (regularly planned process points within the project) the size estimates are updated and the productivity driver assumptions are revisited. For example, the initial parameter settings with respect to tools, the level of testing, and the level of documentation will be revisited to see if the parametric model can be better calibrated to actual performance. Scheduling assumptions should be updated to reflect the actual performance.

The calibrated model may indicate a final cost and schedule that is not acceptable within the defined project goals. If that occurs, the application of the control process can be applied to evaluate the ROI for any changes that can be implemented into the process in order to improve performance against project goals.

Optimal Project Control

As defined previously, optimization of any activity requires the identification of a "cost function." With a parametric model in place to support analysis of project alternatives, a series of stages of the project will be defined. For example, stages might be defined on regular calendar intervals such as bi-annually, quarterly, or monthly depending on the size and planned duration of the project. Alternative stages could be defined at specific planned reviews such as preliminary design review or detailed design review.

Metrics are collected at each stage of the process and utilized in the "baseline" project model to provide a performance based

EVM projection of the estimated cost to complete. The model is then used to evaluate defined alternative management actions against the project optimization cost function. The "best" corrective action is chosen and applied. With revised parameters defined the model is updated to correspond with the time at the stage and a revised project plan with the incorporated project control changes is developed [5].

This process has been applied in multiple instances for clients in the field. A simple example software program for an unmanned missile command-and-control function has been used to provide a demonstration of the concepts. With that application, the project was approximately 48K SLOC of reuse and new code with a resultant 36K of Effort SLOC. The cost function was defined so as to minimize the difference between the actual project effort and the budgeted project effort. At the quarterly review the metrics to indicated completion of systems requirements and about 90% of the software requirements, which should have been completed. The Cost Performance Index (CPI) was 0.96. SEER-SEM with the four dimensional earned value capability of the project management control provided the EAC for both effort and schedule. Using the parametric model of the project the actual metrics were input as a "snapshot" of the project with an estimate of an increase in effort against budget at completion of approximately 3,000 effort hours and 1.2 schedule months. The model was utilized to evaluate the impact of alternate management actions. It was determined the best ROI would result from the change to a development system that minimized the rehost of software from the development to the target system. Assuming that change and updating the model to reflect that revised management input, the program was continued with the result that at the next quarterly review the CPI indicated a .98.

Dynamic Programming

In order to establish the effective best control changes, the theory of dynamic programming was utilized. Dynamic programming is a method of solving problems that exhibit the properties of overlapping sub-problems and optimal substructure [9]. By applying the dynamic programming paradigm it can be shown that by optimally selecting sub-projects at each stage of a project we can be assured of an optimal (or best) result for the total project. As applied to our management control of software projects, the theory validates that we will have a "best solution" by:

1. **Establishing and implementing a valid project plan.**
2. **Monitoring results on a regular basis (i.e. utilize metrics to evaluate performance against the project goal or cost function).**
3. **Define corrective management actions when indicated by the metrics analysis.**
4. **Evaluate the corrective actions to determine a "best" change in performance of the process.**
5. **Repeat the actions at the next stage.**

Following the process will result in a "best" project performance.

Summary and Conclusions

The use of parametric models is well accepted for software project estimation and planning. Maintaining the models at a high fidelity level for the duration of a project has been demonstrated to provide high confidence estimates of expected cost to complete and optimized selection of corrective actions. Updated models when calibrated to the actual performance provide a confidence level estimate of future performance based on results to date. The updated models generate stoplight charts and a set of information similar to EVM outputs but based on project estimates from a calibrated model.

Cost/effort parametric estimating models provide a powerful aid to project management, yielding "objective information" for systems tradeoffs, project management decisions, and controlling project performance throughout the program's lifecycle.

ABOUT THE AUTHOR

Dr. Denton Tarbet is a Senior Consultant with Galorath Incorporated. His research area at the University of Houston was Optimum Control Theory and he has more than 40 years experience in systems and software engineering, successfully managing projects with management responsibilities as Director of Engineering, Chief Technical Officer, and Vice President of Engineering. He has performed size and effort estimation projects for NASA, Air Force, Navy, Republic of South Korea, and multiple contractors.

E-mail: dtarbet@galorath.com
Phone: 310-414-3222

REFERENCES

1. Web site: <http://www.irise.com/blog/>
2. Conference Proceedings: Ross, M, Parametric Project Monitoring and Control – Performance-Based Progress Assessment and Prediction, SCEA, April 2005
3. Journal: Tarbet, D., On Time and On Budget, as Specified: An integrated approach to the Software Development Lifecycle, The Rational Edge, IBM Jan. 2008.
4. Journal: Smith, L. Overview of Project Management, STC Crosstalk, Jan 2003
5. Conference Proceedings: Tarbet, D, Software Project Management – Controlling the Process, PSM User's Conference, Vail, CO, July, 2007
6. Conference Proceedings: Madachy, R and Tarbet, D, Case Studies in Software Process Modeling with System Dynamics, Software Process Improvement and Practice, 5(2-3), 2000 (Initial version in Proceedings of ProSim Workshop 1999.)
7. Book: Galorath, D and Evans, M, Software Sizing, Estimation, and Risk Management, Auerbach Publications, Boca Raton, FL, 2006.
8. Conference Proceedings: Tarbet, D, Project Management Decisions Based on SW Metrics – Modeling Beyond the Estimate, SSCAG Software workshop, Santa Monica, CA Jan., 2006
9. Book: Bellman, Richard, Applied Dynamic Programming, Princeton University Press, Princeton, N.J., 1962

Business Process Management Field Guide

D.B. Robi, Lockheed Martin

Abstract. As more technologies such as SOA, web services, collaboration, social networking, and handheld devices are adopted and deployed, typically through IT departments, the full benefits and return on the investments cannot be realized until the supported business processes are updated and modified to leverage and exploit these new technologies. This Business Process Management (BPM) field guide is a high-level practitioner's guide to performing BPM. The guide takes a very practical approach to BPM illustrating techniques and supporting artifacts that will enable improvements and optimization of the business process relative to available technologies. The intent is to provide a means for the non-IT businessperson to be able to recognize the need, promote, and perform BPM within their own organizations.

1. Overview

This field guide takes a very practical view of BPM focusing on three simple but comprehensive steps: document, align, and optimize.

The underlying premise driving the need to perform BPM is that the full benefits and return on the investments of moving to new technologies cannot be realized until the underlying business processes are updated and modified to leverage and exploit the new technologies. The example I like to use is, if your business had a communication problem it could not be solved by simply providing all employees with cell phones. To truly solve the communications problem, changes to policies and processes would have to be made to leverage and exploit the new cell phone technology.

These three steps—document, align, and optimize—may be performed serially, focusing on one specific business process or may be performed in parallel for larger projects. The initial step, document, focuses on identifying, characterizing, and capturing the business process. The document step develops a model of the business process enabling analysis and refinement. It is most crucial to first define and understand the process, including its relationships and dependencies, before making any modifications or improvements. Once documented, the process can be aligned. The alignment step targets how the process is implemented and supported. The tools and technologies supporting the process are identified along with the required underlying hardware and infrastructure. Once the process is documented and aligned it can be optimized. The optimize step takes a variety of approaches from simple technology upgrades to full invasive process modification and improvement. The depth of the optimize step is dependent on the business significance of the process. Some processes are documented, aligned, and done; others will require and warrant full optimization. These decisions will be driven based on the outcomes and findings from the document and align steps.

This approach was developed and refined over the course of several years on a variety of projects in both the federal/DoD and private/commercial sectors. These engagements typically lasted eight to 12 weeks and focused on business processes in areas including: bid and proposal, finance and accounting, human resources, hotel and restaurant services, auditing and reporting, electronic purchasing, resource allocation process, and strategic planning.

This white paper will describe and define each of these three steps: document, align, and optimize, in terms of activities and products. Additionally, based on experience, one more initial, yet very crucial step, motivation must be added. Before embarking on any BPM project, management must first be convinced that there will be a pay off in order to gain approval and funding. Our discussion starts by providing some definitions and concepts to establish an overall context.

2. BPM Concepts

BPM is defined as a set of integrated, closed-loop management and analytic processes, supported by technology that addresses financial as well as operational activities. BPM is an enabler for businesses, defining strategic goals, and then measuring and managing performance against those goals [1].

BPM encompasses a broad set of capabilities and features including business process discovery, design, modeling, simulation, deployment, execution, administration, control, management, and optimization. Some of the key BPM attributes are:

- Alignment of overall organizational strategies with business objectives and supporting processes and technologies.
- Development of an enterprise process model that provides an end-to-end view of core processes.
- Performance management based on business driven metrics and measures.
- Business agility, accountability, and visibility.
- Robust process improvement methodologies including Lean, Six Sigma, Continuous Improvement, ISO assessments, AFSO21 [2], LM LM21 [3], and quality management systems to facilitate the transition from design to execution and help mature the process lifecycle.

Simply stated, BPM may be divided into two major phases: process development and process monitoring. Document, align, and optimize comprises the process development phase. Process monitoring, commonly referred to as Business Activity Monitoring, focuses on identifying and monitoring Key Performance Indicators (KPI) to determine process effectiveness, efficiency, and overall operational health.

3. Business Process Characteristics

To clarify our BPM approach we need to provide some definitions and descriptions for the terms and objects we will be dealing with and manipulating, starting with a business process.

A business process is "a collection of activities that take one or more kinds of input and creates an output that is of value to the customer" [4].

This is a reasonable definition but not descriptive enough for our purposes so we will leverage the Zachman Framework [5], applying the six interrogatives to business processes; "how" are things performed, "who" is performing those things, "what" information is used, "when" and "where" does it occur and, "why" is it done?

It should be noted that the last question, "Why is it done?" referencing the process itself may be the most significant. Before embarking on any improvement activities, we always need to ensure that the business process being reviewed is still relevant and needed. Experience indicates that all processes do not meet these criteria and furthermore once a process is defined and in use, our natural resistance to change makes it very difficult to remove or modify it.

Business processes may be viewed as containers for the business know-how of organizations; they are the value delivery systems for the enterprise. Business processes cut across functional organizational boundaries. An enterprise typically has four levels of processes, this is a common concept in industry, but the specific terminology is not industry common. For our purpose here we shall define these four levels to be: core processes, sub-processes, activities, and tasks.

A core-process is initiated in response to a customer request and crosses functional organizations providing value back to the customer. The sub-process typically resides within one functional organization to support a core-process request. The activity level is typically viewed as one step in the flow of a sub-process, this is what we typically model in business process diagrams and the task is one step of an activity typically performed by one person.

Core processes are what the enterprise exposes to customers. A typical enterprise may contain five to 10 core business processes. A banking enterprise would identify core processes such as: deposit to an account, withdraw from an account, and get account balance. These core processes are in turn supported by sub-processes such as: verify customer, and update account. A sub-process, like, verify customer, would be supported by activities like: validate customer name and validate customer PIN. Finally, at the task level you would find the look-up and retrieval tasks used to access the specific customer data elements.

4. Motivation

The initial hurdle to overcome and question to answer before embarking on any BPM task is, "Why do it?" Management typically withholds funding until they are very clear on what the return on their investment will be. This is typically best answered and presented to your management team in terms of quantifiable questions.

Does your business operate in a continual state of crisis?

- If you are always addressing problems and fire fighting, when and how do you have time to strategize and plan?

Is your business unable to keep pace with technology and customer expectations?

- Today's customers are members of the so-called "Nintendo Generation." These are the people regularly using Facebook, YouTube, text messaging, BlackBerries, smartphones and tablets, some of them every minute of every day. They are aware of new technologies and how things can be done better, faster, and cheaper using those technologies.

How outmoded are your business's current set of practices?

- Have your business processes ever been reviewed for possible improvements?
- Have your business and processes grown up following an "urban sprawl" model, with no rhythm, reason, or plan?
- Have any of your processes been developed using built-in latencies that are no longer relevant or needed? For example, the QWERTY keyboard was originally created to slow the typist down due to mechanical short comings.

Answering these questions will illustrate and quantify the need for and motivate the action to perform the BPM task.

5. Document

In order to perform our document, align, and optimize BPM approach we first start by identifying the processes. Once identified, the process will be documented using the Object Management Group's (OMG) standard Business Process Modeling and Notation (BPMN) [6] graphical notation. I will address each of these: identify and document steps separately, as the identify is typically the more difficult.

Identifying business processes, more commonly referred to as process discovery is simple in concept: describe your company's key processes. Gartner indicated that process discovery might be the most difficult and important skill of all "the most critical practice to master is the initial discovery and definition procedure for the target process [7]." Our approach combines Hammer and Champy's "Reengineering the Corporation" [8] with J. Womack's "Lean Thinking" [9] techniques focusing on the customers, stakeholders, and users and the services and processes they are concerned with.

Starting at the core process level, we identify the customer-facing processes and then the supporting sub-processes. Keep in mind that the core processes cross-organizational boundaries and

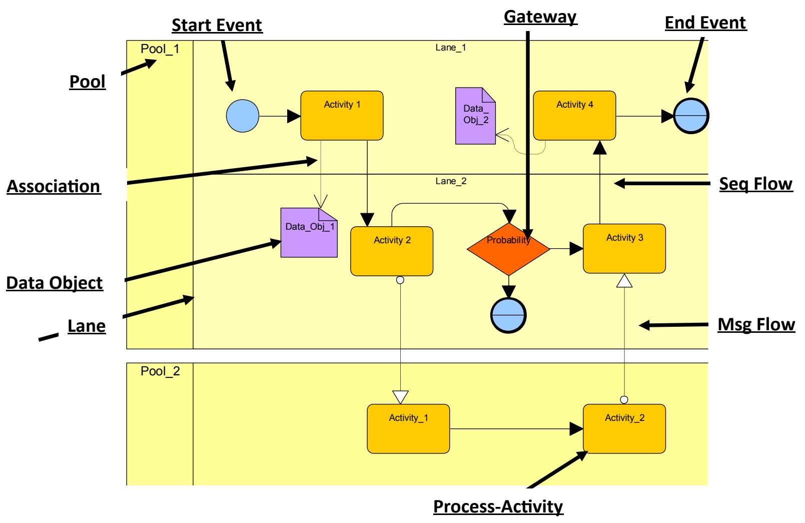


Figure 1 BPMN Sample

sub-processes function within one organization. Some examples of core processes are: product development process, procurement process, market planning process, insurance claim processing, and government proposal process. You will typically model both core and sub-process levels in order to develop a holistic enterprise-level view of the processes.

Process discovery is one technique employed during the document step. Business process discovery may be accomplished by decomposition, that is the partitioning of the as-is processes into architectural significant business processes that are unique and required. Another approach is customer focus, focusing on how external customers obtain goods and/or services from the enterprise. The outcome of these approaches is a set of core and sub-business processes. The core processes are documented in a core business workflow similar to a value stream map using BPMN and textual process narratives. The resulting process narratives or process metadata includes key information about the process including ownership, users, activities, responsibilities, problems, products, outcomes, and relationships to the business's mission objectives.

Once the processes are identified we can graphically document them capturing the process using BPMN (see Figure 1). The BPMN models contain activities and tasks, relative to: order of operations, products (input and output) organizational participation (roles and responsibility), events (business rhythms), and scope (start and endpoints). These BPMN business process models become the starting point for all changes promoting adaptability and agility to quickly see and understand the impacts and implications of business process changes. A technique employed to support the alignment phase is to include, as swim lanes, tools, and technology. This enables the association of sub-processes and activities to their supporting COTS software and hardware. Business processes are graphically documented using standard OMG BPMN supported by an enterprise architecture COTS modeling tool such as: IBM Rational System Architect, TROUX Métis, or Microsoft Visio 2010, the document, align, and optimize BPM approach, methods and techniques are tool agnostic.

As-is to To-be Modeling Approach

The initial process model is the so-called as-is model and is intentionally sparse with regard to process documentation because our approach is to-be focused. We define the as-is model to the level of detail needed to expose and understand the issues and shortcomings of the existing process. Based on our experience in performing BPM tasks, we have found this to-be focused approach appropriate and necessary in order to move people from the comfort of the existing as-is world ahead into the new to-be world. The as-is model provides a baseline to begin the in depth analysis needed to develop and document the improved to-be process.

Note: BPMN provides a standard set of diagramming conventions for describing business processes and is intended to support the capture of sufficient details to allow the documented process to be the source of an executable process description. BPMN is to be the graphical front-end to Business Process Execution Language (BPEL). BPEL process descriptions enable automated process execution; automated BPMN to BPEL conversion is still a work in progress. The current industry wisdom is to use tools that generate standard OMG BPMN to position your business to leverage automated BPEL generation products when available and mature.

6. Align

Our BPM approach takes a holistic enterprise view. Our view of an enterprise is a collection of business systems or a "system of systems." These systems control and manage the enterprise's functional areas, generically: facilities, regulations, operations, procurement, human resources, finance, and supply (see Figure 2). Many of today's business systems were developed as disparate sets of applications, these are the existing "as-is" legacy business applications typically described as "functional silos" or "stovepipes." These disparate applications are the result of years of development without a central vision or strategy (no architecture). Applications have grown up that serve a single purpose for a single set of users, with no thought to integration. Today, in our enlightened state, we realize that these collections of disparate applications need to be modernized and consolidated into an agile set of unified and integrated business processes. The facilitating discipline to achieve this is enterprise architecture.

An enterprise's competitive edge and ultimate success are enabled by the ability to rapidly respond to changing business strategies, governances, and technology. This competitive edge translates into higher levels of customer satisfaction, shorter work cycles, and reductions in schedules, maintenance costs, and development time, all resulting in lower overall costs of ownership. Enterprise architecture is the key-facilitating ingredient, providing a holistic view and a mechanism for enabling the design and development, as well as the communication and understanding of the enterprise. The overarching goals of enterprise architecture are to manage the complexity of the enterprise, align business strategies and implementations, and facilitate rapid change in order to maintain business and technical advantages.

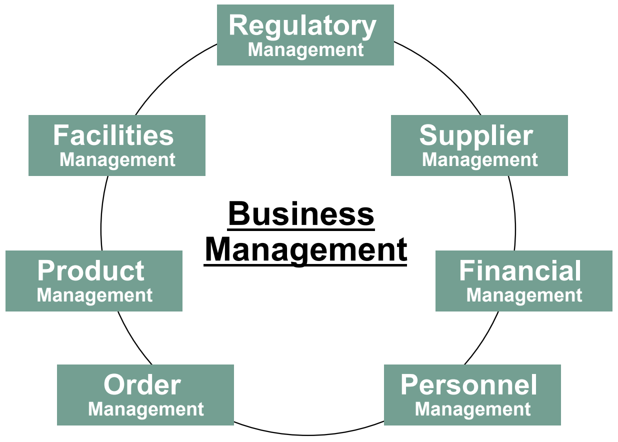


Figure 2 Abstract Business Processes

Our Enterprise Model

How is the enterprise aligned? The enterprise is divided into four views (dimensions): value, strategy, process, and technology (see Figure 3). The value view contains the enterprise value proposition. The goods and services produced by the enterprise that customers are willing to pay for. The Strategy view includes: the business drivers, strategies, goals and objectives. This is defined in the business charter and mission statement along with the business strategic plan. The process view describes the core business functions, supporting processes and enabling activities. Finally, the technology view describes the applications, reusable components (commercial and developed products) as well as the integration and technology services (middleware, security, communications) and the supporting infrastructure.

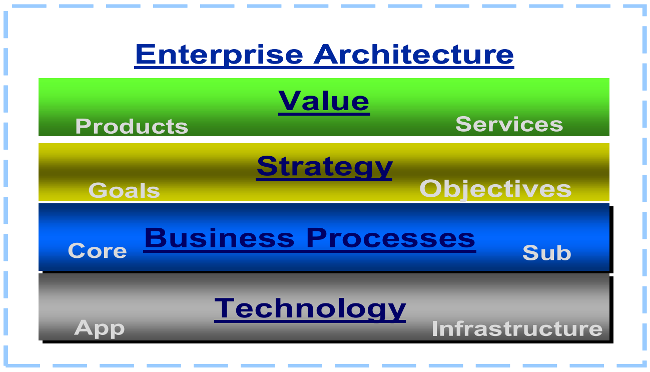


Figure 3 Abstract Enterprise Model

Process alignment occurs up and down the model. Each core process should be aligned to directly support the enterprise's value and strategy. Example: If a banking enterprise, the core processes should support the featured banking products like savings accounts and student loans. If a product supplier, the core processes should support the activities of order fulfillment and billing. The core business processes should directly support and positively contribute to the enterprise's value proposition and strategic plan. (E.g. Wal-Mart's inventory management or FedEx' package tracking not human resources.)

Similarly, the business processes must be implemented and supported in the most effective manner. The processes need to be aligned to the underlying supporting technology to demonstrate that the one best technical product/tool is being used to implement the processes. The process to technology alignment facilitates reliable process execution by ensuring that the processes are being implemented consistently using the same underlying technology and tools. In our experience, this is not always the case. As enterprises grow, via mergers and acquisitions, there are typically multiple technologies being used to perform the same process resulting in process variations.

The alignment step can complete the BPM cycle as long as the process in question is consistently and efficiently functioning and providing the desired outcomes. This is true even if technical re-alignment is required, because swapping out technology does not affect the internal workings of the process. The optimize step is taken when the process itself is broken and requires internal changes.

7. Optimize

The optimization step starts with the as-is process documented in BPMN. Our view of optimization is a continuum from technical to functional. The technical end is the most basic, focusing on simple, narrow scope changes which may include upgrading screens or replacing functionality with a COTS package. The technical end starts by targeting the so-called "low-hanging fruit" to provide quick tangible working results. At the other end of the continuum is functional where we consider a long term, broad view. Functional modernization is invasive, performing process modeling and optimization as required, typically resulting in major changes to processes and associated technologies.

Typically, technical modernization realizes medium, and immediate gains in lowering the total cost of ownership and improving the Quality of Service (QoS). These technical changes will plateau moving the project into the functional modernization phase targeting the business processes and sub-processes for additional improvements in costs and quality.

If we recall, that core customer facing business processes cross organizational boundaries, then it follows that these processes cannot be optimized organizationally, but must be viewed from an enterprise perspective. We need to take a process-oriented approach. Start by identifying who uses the process, these are the process customers, and then identify the things, goods and services provided. This process oriented approach shares similarities to Object Oriented (OO) methodology. Always keep in mind, that we are focusing on the people using the process and the value added things (objects) being produced not the specific activities (procedures); again similar to an OO approach.

Our experience indicates there are several recurring process improvement themes that typically come to play during the course of process optimization:

• **Team Approach:** This situation may present in the form of needing to perform numerous hand-offs during the course of executing the process. Hand-offs are typically problematic especially between departments or organizations. This may be resolved by taking a team or caseworker approach and empowering the team to make required decisions. Also, certain sub-processes naturally occur in order to complete a whole piece of work.

An example may be a typical bid and proposal process. One team designs the technical solution; another team determines a price to win. Once these two activities are complete, they are reconciled. Why not cross-pollinate the teams by combining technical experts with financial experts, allowing the team to self-correct as they proceed, and eliminating the need for after the fact reconciliation. By combining these sub-processes into the same organization we can again minimize hand-offs. The key is to limit the span of information passing.

• **Minimize Reconciliations:** There are many processes that are characterized by external dependencies. Typically, these dependencies are only reviewed at milestone points. If these dependencies are not in tune, then there must be a reconciliation and determination of what and who needs to change. By building checkpoints into the process, rather than waiting for formal reviews, we can mistake proof our processes with regard to these external dependencies. This may be combined with the team approach.

• **Process Visibility:** Based on performing numerous process improvement tasks, spanning 25 years, this issue is the granddaddy of them all. Process visibility, knowing your status, where are you in terms of the process, what is your current position, these questions and resulting issues are almost always present. Experience has demonstrated numerous instances where the process is functioning as it should, but the Users are not aware of it due to lack of visibility. The ability to provide self-service process status is critical to establishing process confidence. Included here are the user's timing issues, when will the order arrive, has it been shipped, what reviews have taken place and what reviews still need to occur. These are all process events that users are very interested in. Methods used to provide process visibility have included providing electronic dashboards down to hand written placards; the common feature is providing insights into the process or process visibility.

• **Listen:** Here again experience has made this point abundantly clear; simply listening to our customers or users is pure gold. Unfortunately, this skill is underutilized and too often process improvement practitioners will only listen long enough to formulate a conclusion relative to a known solution. In most cases, the proposed solution will not be accepted by the customer, even if the solution is correct, due to the lack of the customer's trust in a quick turn solution.

The technique known as "active listening" should be applied (see Figure 4). What is active listening [10]? It is paying close attention to what the customer is saying and providing both verbal and visual feedback indicating that you are hearing what is being said, that you are in fact hanging on every word. The customer, as with most people, wants to be heard. This simple ability to listen has very positive and significant outcomes including: truly understanding the problem, demonstrating your sincerity to help, and most significant (again from experience) hearing the solution. Meeting and listening to your customer will help identify their points of pain, where does it hurt, what is not working well and what is working well. In my experience, I have found that the people who can solve the problem are the people executing the process, they are the closest to the process and typically know what is wrong and how to fix it, but you have to listen.

Active Listening Characteristics:

- Look at the person, and suspend other things you are doing. (Physically moving yout chair helps)
- Listen not merely to the words, but the feeling content.
- Be sincerely interested in what the other person is talking about.
- Restate what the person said.
- Ask clarification questions.
- Be aware of your own feelings and strong opinions.
- If you have to state your views, state them only after you have listened.

Figure 4 Active Listening Approaches

8. Declaring Success: Measures and Metrics

Following the strategy of Dr. W. Edwards Deming, "We cannot improve what we cannot measure" [11], we need to evaluate and quantify business process improvements by identifying the appropriate metrics and measurements.

Performance measurements and metrics provide the means to evaluate process execution. We typically identify three types of metrics across an organization: strategic, performance, and operational.

- Strategic metrics provide a direct link to the strategy and goals of an organization, are aimed at executive management or shareholders and usually include financial measures.
- Performance metrics decompose the strategic metrics to a lower level aimed at measuring a process' business contribution including customer satisfaction and productivity.
- Operational metrics measure the day-to-day business operations providing insight on how the business is actually running, include process cycle times, availability, reliability, and overall QoS.

A balance of these metrics provides the most complete and accurate picture of the health of the organization. Metrics are indicators of business capabilities and measures are a means to derive or calculate and expose the metrics. As an example, for Major League Baseball the hitting ability metric is the batting average and is measured using at bats and hits. The measures in business are defined as KPIs that align to specific business processes and activities in order to determine overall process health. Once these KPIs are identified, driving metrics can be designed into an executive business dashboard, providing quick insights into a process' strategic performance and operational health. Business processes need to be measured in terms of the enterprise and aligned with the organizational goals and objectives.

9. Summary

This article starts with the premise that the full benefits and return on the investments of moving to new technologies cannot be realized until the underlying business processes are updated and modified to leverage and exploit the new technologies. BPM is the means to accomplish these process improvements and our document, align, and optimize approach provides a practical and simple method. The document step focuses on identifying and graphically describing the as-is process using standard BPMN to develop the as-is model enabling process analysis and refinement. Once documented, the process can be aligned. The alignment step focuses on the uniform application of technology insuring consistent implementation producing reliable outcomes. The tools and technologies supporting the process are identified along with the required underlying hardware and infrastructure. Some processes are documented, aligned and done; others will require and warrant full optimization. These decisions will be driven based on the outcomes and findings from the document and align steps. Once the process is documented and aligned it can be optimized if warranted. The optimize step takes a variety of approaches from simple technology upgrades to full invasive process modification and improvements. The depth of the optimize step is dependent on the needs of the business, driven by the projected savings from developing the improved to-be process. To these three steps an initial step was added, motivation, used to provide management the benefits and projected savings of performing this BPM project.

A final note; realize that moving into someone else's business or house and performing BPM is like cooking Thanksgiving dinner while remodeling the kitchen—the business cannot shut down. Common wisdom is to empathize with your customers and users and make every attempt to make this experience as painless as possible. Good luck.

ABOUT THE AUTHOR

D. B. Robi is a Lockheed Martin Qualified IS&GS Information System Architect in the area of Business Process Transformation and is a Certified Lean Six Sigma Black Belt. His career, spanning 27 years, has afforded him opportunities to work in both the commercial and DoD/Federal sectors. He has performed as Solution Architect, Chief Engineer, and Enterprise Architect on numerous modernization projects including projects involving large commercial corporations as well as serving as the Business Architect on efforts in Business Process Modernization, Reengineering, and Optimization. Mr. Robi is a steering committee member for the IS&GS Business Process Transformation Community of Practice and also consults on many internal LM projects. Mr. Robi was a major contributor to the development and enhancement of Lockheed Martin's, internally developed, ARQuest® Blueprint and OMEGA® BPM, approaches to developing enterprise architectures, business transformation and transition plans. He has developed, published and presented many papers on technical topics including: BPM, SOA, Requirements, Use Cases, Enterprise Architecture, Value Analysis, and Architectural Modeling. He developed and published the "Motivational Views", extensions to DoDAF, and incorporated and implemented this concept in Lockheed's ARQuest product.

Lockheed Martin Information Systems & Global Solutions
Owego, NY 13827
Phone: 607-751-5358
E-mail: dennis.robi@lmco.com

REFERENCES

1. The BPM Standards Group: Applix, Hyperion, IBM, META Group, SAP AG and The Data Warehousing Institute
2. Air Force Smart Operations for the 21st Century
3. Lockheed Martin in the 21st century
4. Hammer, Michael and Champy, James Reengineering the Corporation: A Manifesto for Business Revolution, New York: Harper Business, 1993
5. Zachman, John A. "My Framework", IBM System Journal, May 1967:125. <www.research.ibm.com/journal/sj/382/zachman.pdf>
6. Business Process Modeling Notation, Version 2.0 Object Management Group, Jan 2011. <http://www.omg.org/spec/BPMN/2.0/PDF>
7. Melenovsky, Michael J. "Business Process Management's Success Hinges on Business-Led Initiatives", G00129411 Gartner (July 2005): 26
8. Hammer, Michael and Champy, James, op cit.
9. Womack, James P. and Jones, Daniel T. Lean Thinking Womack. New York: Simon & Schuster, 1996.
10. Active Listening. <http://en.wikipedia.org/wiki/Active_listening>
11. Deming, W. E. Out of the Crisis, Cambridge, MA: Massachusetts Institute of Technology Press, August 2000.

ADDITIONAL READINGS

1. Harmon, Paul. Business Process Change: A Manager's Guide to Improving, Redesigning, and Automating Processes. San Francisco: Morgan Kaufmann Publishers: 2003 [ISBN: 1-55860-758-7]
2. Robi. D. B. BPM Enabled by SOA. Lockheed Martin Mission Critical Enterprise Systems Symposium. Orlando, 2007
3. Long, Kathy. "BPM and Process Modeling", Process Renewal Group, <www.processrenewal.com>
4. Harmon, Paul. The Evolution of Business Process Management. Business Process Trends DCI BPM. New Orleans, 2005. <http://www.bptrends.com>
5. AFSO21 Playbook- <http://www.af.mil/shared/media/document/AFD-070205-088.pdf>
6. Joyce, Michael and Schechter, Bettina. "The Lean Enterprise- A Management Philosophy at Lockheed Martin." Defense Acquisition Review Journal, (2004) <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA435279&Location=U2&doc=GetTRDoc.pdf>
7. Robi, D. B. "Enterprise DoD Architecture Framework and the Motivational View." CrossTalk, (April 2004) <http://www.stsc.hill.af.mil/crosstalk/2004/04/0404Robi.html>
8. Kirkpatrick, D. "Shaping the Process – Managed Organization of the Future", <http://www.thecelestiagroup.com>
9. Ashton, Heather and Kelly, David. "The Business Impact of BPM with SOA: Building a Business Case for BPM with SOA ROI", Upside Research Inc, 2006. <www.upsideresearch.com>
10. Chappell, David, Understanding Service-Oriented Architecture (SOA) and Business Process Management (BPM). New York: Chappell and Associates, 2005 <www.davidchappell.com>
11. BEA System Inc, "Extending the Business Value of SOA Through Business Process Management." 2006, <http://www.ebizq.net/web_resources/whitepapers/BPM-SOA_wp.pdf>
12. Lombardi Software."Process Discovery-The First Step of BPM", 2007 <www.lombardi.com>
13. Vollmer, Ken. "What is the Relationship Between BPM & SOA and Why Should You Care?", Forrester Research 2006, <www.forrester.com>

Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

Software ID Summit
2 May 2012
Campbell, CA
http://tagvault.org/2012-summit

Security and Software Engineering Research Center (S2ERC) Showcase
15-17 May 2012
Arlington, VA
http://www.cyber.vt.edu/s2erc

SEPG Europe 2012
5-7 June 2012
Madrid, Spain
http://www.sei.cmu.edu/sepg/europe/2012

Software Assurance Working Groups - Summer 2012
26-28 June 2012
McLean, VA
https://buildsecurityin.us-cert.gov/bsi/events.html

INCOSE International Symposium 2012
9-12 July 2012
Roma, Italy
http://www.incose.org/newsevents/events/details.aspx?id=142

Practical Software and Systems Measurement (PSM)
16 July 2012
Mystic, CT.
http://psmsc.com/Events.asp

COMPSEC 2012
16-20 July 2012
Izmir, Turkey
http://compsac.cs.iastate.edu/

GFIRST8
19-24 August 2012
Atlanta, GA
http://www.us-cert.gov/GFIRST

Events continued...

26th International Biometrics Conference
26-28 August 2012
Kobe, Japan
http://www.ourglocal.com/event/?eventid=11988

Diminishing Manufacturing Sources and Material Shortages & Standardization
27-30 August 2012
New Orleans, LA
http://www.dmsms2012.com/

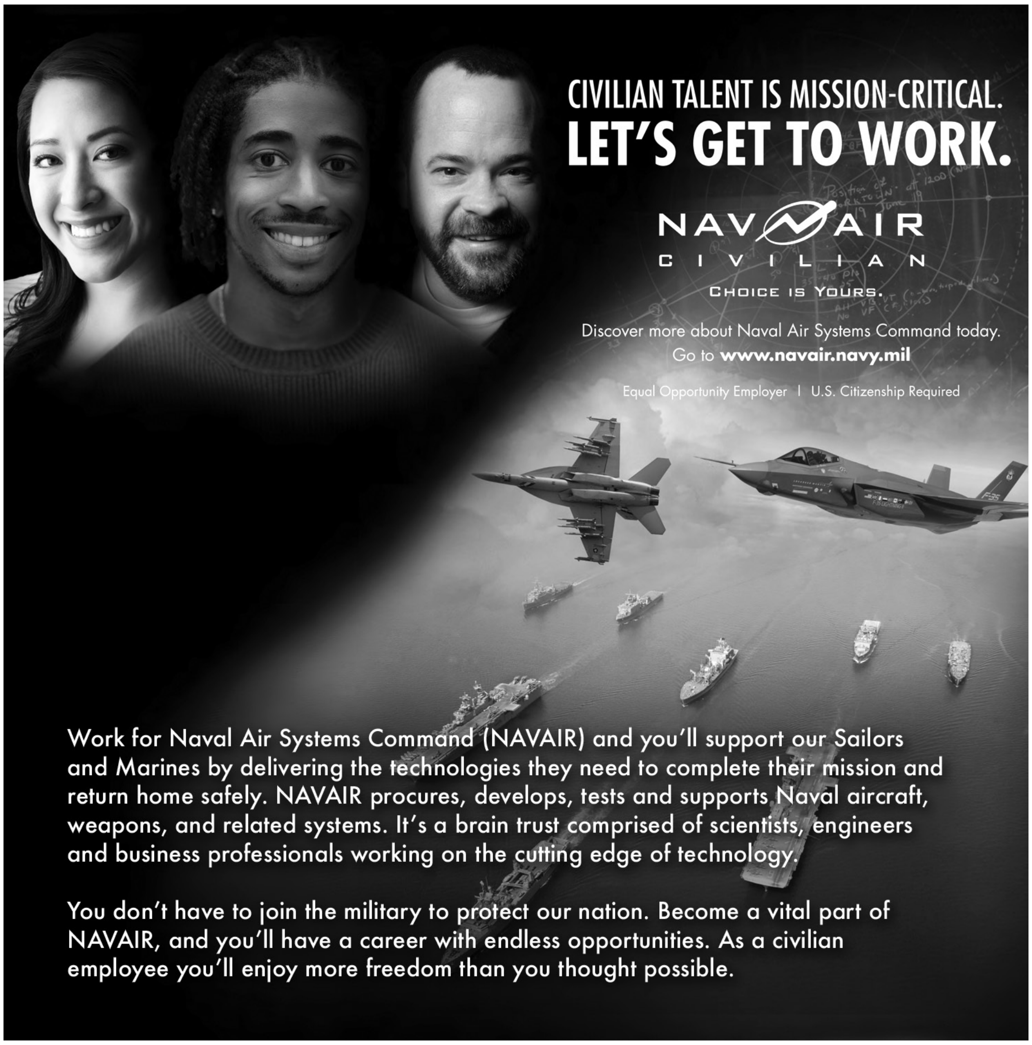
AUTOTESTCON 2012
10-13 September 2012
Anaheim, CA
http://www.autotestcon.com/general/autotestcon-2012

ASIS/(ISC)2 Security Congress
10-13 September 2012
Philadelphia, PA
https://www.isc2.org/congress2012/default.aspx

15th Annual Systems Engineering Conference
22-25 October 2012
San Diego, CA
http://www.ndia.org/meetings/3870/Pages/default.aspx

OWASP AppSec USA 2012
22-26 October 2012
Austin, TX
https://www.owasp.org/index.php/Category:OWASP_App-Sec_Conference

12th Annual CMMI Technology Conference
5-8 November 2012
Denver, CO
http://www.ndia.org/meetings/3110/Pages/default.aspx


CIVILIAN TALENT IS MISSION-CRITICAL.
LET'S GET TO WORK.
NAVAIR
CIVILIAN
CHOICE IS YOURS.
Discover more about Naval Air Systems Command today.
Go to www.navair.navy.mil
Equal Opportunity Employer | U.S. Citizenship Required

Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.

You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.

24th Annual

SSTC

Systems & Software Technology Conference

Thank you to all of the sponsors, exhibitors, and attendees that helped make the 24th Annual Systems & Software Technology Conference a big success in April 2012! Visit www.sstc-online.org or follow our Facebook page for information about next year's event.

WWW.SSTC-ONLINE.ORG

Follow Us On Facebook
http://www.facebook.com/TheSSTC

WAR FIGHTING TECHNOLOGIES

ENHANCE ADVANCE MODERNIZE

Instability and Faith in the Art of Computer Programming

I confess—I have a preference for a certain electronic tablet and smartphone. I am not going to give its name, but suffice it to say that over the years I have learned to rely on it for more uses that I thought possible. It comes with 3G connectivity so it allows me instant access to the Internet almost anywhere. Once you start using one, well, it becomes part of your life.

Need an address? Contact list. Need a location? Maps. Want to see who was the stunt double for Judy Garland in "The Wizard of Oz?" Use IMDB. (It was Bobbie Koshay. And, for you true Oz fans—yes, I know—Caren Marsh-Doll was the off-camera stand-in, but NOT her stunt double.)

The problem is, you begin to depend upon these mental crutches. I no longer remember phone numbers—why bother? Let me use these wonderful appliances to push back the effects of senility and dementia just a little bit longer. I rely on these mental crutches so much so that my daughter refers to it as my, "second brain."

Until, of course, they do not work. I am not talking about the typical, "I am out of range, let me hope I find a signal soon." I am talking about the feeling I recently had when an update to the software on both my phone and tablet caused my formerly rock-solid browser to crash. Frequently. Every few minutes. All the time. Constantly. All the time.

While at my parents' house recently, the only Internet access available was my 3G tablet/phone. However, I was unable to reserve a hotel room for the return drive. On an earlier trip—I could not book an airline ticket. I was not even able to enjoy my morning ritual—coffee and browsing the Internet.

Now, I am not saying that my life was a catastrophe. I am just saying that instead of depending upon connectivity and modern technology, I (GASP!) had to make phone calls. Talk to real people. Read and touch an actual newspaper. And—when a recent update to the software fixed the problems—I somehow still seem to still have a bit of mistrust in my connectivity issues. It is just a little bit harder to take it for granted. I have been burned, and sometimes burns take a while to heal.

Which got me thinking. We have had a few depressions/recessions in the past 100 years. And if you talk to an economist, they will tell you that one underlying cause is consumers who have lost faith in the stock market.

Is this where we are heading in the computing age? I talk to more and more users who complain about lack of faith when using applications.

There are some software applications where we absolutely cannot have lack of faith. Pacemakers. Reactor software. Braking software in cars. Aircraft control software. And the list goes on and on. As more and more large-scale applications are used worldwide, the need for more and more faith in the software becomes a necessity.

I currently teach Software Engineering, along with Requirements Engineering and Systems Design. And, as part of both courses, I cover some really spectacular software failures (see <http://spectrum.ieee.org/computing/software/why-software-fails> for a great list). There is no shortage of cases to discuss. Some with loses of billions of dollars. Some that caused lost of life. Some that caused large companies to simply go out of business. And in almost all of the cases, one of the common factors tends to be lack of user involvement. Some might find it simply amazing that when writing software that might cost millions and millions of dollars there is a lack of user involvement. However, to experienced practitioners of the craft of producing software, it is not amazing at all. Bizarrely, the more software seems to cost, the less concerned users are with involvement in its creation. It is like the mere act of spending lots of money somehow relives them of the responsibility of being…well….responsible. In fact, for really large-scale software, it is often next to impossible to actually identify who the users are!

I have been teaching software engineering since the 1970s. Back in the day, we had the waterfall model. We realized, of course, that nobody really followed it—but it was what we had. That is, until the Spiral Model came along. And we realized that everybody really did it over and over and over again until they got it right (or, if not right, at least usable).

But the large-scale models did not adequately address the constant and continuing needs for more user involvement, getting the requirements right, getting a design that will be usable, uncovering the missing functionality, making sure that users' needs and wants are met, and ensuring the validity of the software before it is released to the users.

Is agile and/or rapid development the cure to all of our ills? Does it guarantee the correct software? Does it ensure validity? Does it make the users happy?

No, of course not. It is just a technique. But, it is a technique that focuses on smaller increments. It also focuses on having user involvement frequently, and in time to gently guide the development process so that it has a better chance of delivering software that the user will have a stake in.

And it will help in delivering software that users might just have a little more faith in.

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu

NAVAIR Vision Statement:

"Sailors and Marines, Armed and Operating with Confidence"

Because we develop, deliver, and sustain aircraft, weapons, and systems—on time, on cost, with proven capability and reliability—so they cost effectively succeed in every mission and return home safely.

NAVAIR Goals:

Current Readiness: Contribute to delivering Naval Aviation Units Ready for Tasking with the right capability, the right reliability and safety, in the fastest possible time, and at the lowest possible cost.

Future Capability: Deliver new aircraft, weapons, and systems on time and within budget that out-pace the threat, provide global reach and persistence, support AIR-SEA Battle, Joint and Coalition Operations, and meet the required adaptability, reliability, safety and total lifecycle costs.

People: To institutionalize a culture of learning, innovation and exemplary leadership that is warfighter focused, motivated and inspired—that leverages diversity, technology, analytics, transparency and accountability for a dynamic, agile and adaptive World Class workforce.

NAVAIR

NAVAIR Process Resource Team (PRT)

(760) 939-6226

CrossTalk thanks the above organizations for providing their support.